UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

DEPARTAMENTO DE INGENIERÍA TELEMÁTICA

PROYECTO FIN DE CARRERA

INGENIERÍA TÉCNICA DE TELECOMUNICACION: SISTEMAS DE TELECOMUNICACIÓN

SIMULACIÓN DE REDES JERÁRQUICAS P2P

Autor: Sandra Marco Espinel

Tutor: Roberto González Sánchez

Director: Isaías Martínez Yelmo

Leganés, Junio de 2012

EL TRIBUNAL

Presidente:

Secretario:

Vocal:

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día ....., Facultad de la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la calificación de:

Fdo. Presidente Fdo. Secretario Fdo. Vocal

Agradecimientos

En primer lugar, quisiera agradecer el apoyo, comprensión y paciencia a mi familia, que siempre ha estado ahí cuando lo he necesitado.

A Ibai, por sus consejos de ingeniero, por su ayuda siempre que puede (y cuando no, también) y por ponerme las pilas cuando me ha hecho falta.

A todos mis amigos, por su apoyo y porque han conseguido alejarme de la universidad cuando me era necesario. A toda la gente del basket que anda desperdigada por ahí, los SPCs siempre son una inyección de energía. No me puedo olvidar de mis compañeros de carrera que han “sufrido” todo esto conmigo y a mis Charlies, sin las que los viernes al medio día habrían sido muy aburridos.

Asimismo, expresar mi agradecimiento a Isaías y Roberto por guiarme en la realización de este proyecto, por su ayuda, dedicación y disponibilidad.

Muchas gracias a todos.

Resumen

La transmisión de contenido multimedia es uno de los servicios más utilizados hoy en día en internet. Cada vez se utilizan menos las tradicionales arquitecturas cliente-servidor y se apuesta por nuevas estructuras de red que proporcionan un uso más eficiente de los recursos disponibles.

Las redes *peer-to-peer* se caracterizan por crear una red lógica sobre una red física ya existente. Aprovechan, administran y optimizan el uso del ancho de banda de los usuarios presentes en la red por medio de la conectividad entre los mismos. De esta forma se obtiene más rendimiento en las conexiones y transferencias que con algunos métodos centralizados convencionales, donde una cantidad relativamente pequeña de servidores provee el total del ancho de banda y recursos compartidos para un servicio o aplicación.

En este proyecto se estudia el rendimiento de un modelo jerárquico de una red overlay estructurada. Para ello se parte de una implementación disponible en el simulador PeerfactSim.KOM de una red Chord y se realizan modificaciones para que adquiera las características de una red jerárquica. Posteriormente se ejecutan una serie de simulaciones y se estudian los resultados obtenidos.

Por último se realiza un estudio comparativo entre dos redes peer-to-peer jerárquicas: Chord y Kademlia.

Abstract

Nowadays, one of the most used services in the Internet is the transmission of multimedia content. Less and less, traditional client-server architectures are used and the tendency is the implementation of new network structures in order to improve the efficiency of the available resources.

Peer-to-peer systems implement an abstract overlay network on the top of a physical network topology. P2P is a distributed application architecture that partitions tasks or workloads among peers. Peers share a portion of its resources - such as processing power, disk storage or network bandwidth -to be made directly available to other network participant, without the need for central coordination by servers. With this model, peers are both suppliers and consumers of resources, in contrast to the traditional client–server model where only servers supply and clients consume.

In this document it is studied the performance of a hierarchical model of a structured overlay network. A flat implementation of a Chord network available in simulator PeerfactSim.KOM is modified as to add all the characteristics and requirements of a hierarchical network. The proposal is validated launching many simulations in order to obtain reliable outcomes.

Finally, the results of two different peer-to-peer networks are compared: Chord and Kademlia.

Índice

Índice de figuras

Índice de tablas

Capítulo I

Introducción

1.1. Introducción

Debido al notable incremento del número de equipos conectados a Internet [1], una mayor capacidad de almacenamiento de éstos y un rápido incremento del ancho de banda disponible, ha surgido una enorme difusión de recursos de todo tipo. Las redes overlay ayudan a un almacenamiento y búsqueda eficaz de toda esta información.

Una red overlay es aquella en la que los terminales que la componen se organizan definiendo una nueva estructura lógica de red superpuesta a la ya existente basada en la interconexión mediante routers IP. Las redes overlay más conocidas son las denominadas peer-to-peer o P2P.

Las redes de ordenadores peer-to-peer son redes que aprovechan, administran y optimizan el uso de banda ancha por medio de la conectividad entre usuarios participantes en ella, obteniendo como resultado mucho más rendimiento en las conexiones y transferencias que con algunos métodos centralizados convencionales, donde una cantidad relativamente pequeña de servidores provee el total de recursos compartidos.

Dichas redes son útiles para muchos propósitos, pero se usan muy a menudo para compartir toda clase de archivos: audio, vídeo, texto, software y datos en cualquier formato digital. Son también comúnmente usadas en

telefonía VoIP para hacer más eficiente la transmisión de datos en tiempo real, así como lograr una mejor distribución del tráfico telefónico.

Existen diferentes tipos de redes peer-to-peer y distintas aplicaciones han hecho uso de ellas. Cada una tiene sus ventajas e inconvenientes y dependiendo del escenario en que se usen, unas serán más adecuadas que otras. Sin embargo, diferentes redes peer-to-peer no pueden compartir recursos entre ellas debido a sus distintas topologías y mecanismos de enrutamiento.

El objetivo principal de este proyecto es modificar una implementación ya existente del protocolo de red Chord para que se comporte de modo jerárquico y así suponga un primer paso para conseguir el intercambio de información entre redes de distintas topologías y distintos protocolos. Se comprobarán los beneficios de este tipo de redes debido a la reducción del número de saltos necesarios para encontrar la información requerida.

Tras conseguir la implementación de la red jerárquica en el simulador PeerfactSim.KOM se llevarán a cabo una serie de simulaciones para evaluar sus prestaciones. Los resultados se someterán a un estudio comparativo con los resultados obtenidos en otro proyecto realizado en la universidad por Roberto González Sánchez, utilizando otro tipo de red P2P: Kademlia. De esta forma se podrá decidir qué estructura es la más adecuada dependiendo de determinados factores a tener en cuenta de los que se hablará más adelante.

1.2. Contenidos de la memoria

La memoria se divide en diferentes capítulos:

- El Capítulo I se compone una breve **Introducción** a las redes P2P, así como la motivación y los objetivos de este proyecto. Se expondrán algunas posibles aplicaciones y se explicarán de forma esquemática la distribución de contenido de este documento.

- El Capítulo II da una visión general del **Estado del Arte** dando a conocer el estado actual del campo de las redes P2P.

- El Capítulo III se basa en el **Diseño e Implementación** de la red jerárquica Chord.

- El Capítulo IV realiza la **Evaluación** de la red llevada a cabo tras la obtención de los resultados de las simulaciones ejecutadas.

- En el Capítulo V se hace una **Comparación** de los resultados obtenidos con una evaluación de una red jerárquica Kademlia realizada previamente en la Universidad.

- Para terminar, en el Capítulo VI se exponen las **Conclusiones y los trabajos futuros.**

Capítulo II

2. Estado del arte

2.1. Redes Peer-to-Peer

Una red peer-to-peer [2], es un conjunto de equipos conectados por medio de un método de transporte de datos en el que se comparten recursos y en el que no están definidos ni clientes ni servidores fijos. Se puede decir que los equipos conectados a la red se comportan, simultáneamente como clientes y como servidores con respecto al resto y todos tienen capacidades y responsabilidades equivalentes. La figura 1 muestra un ejemplo de arquitectura P2P.

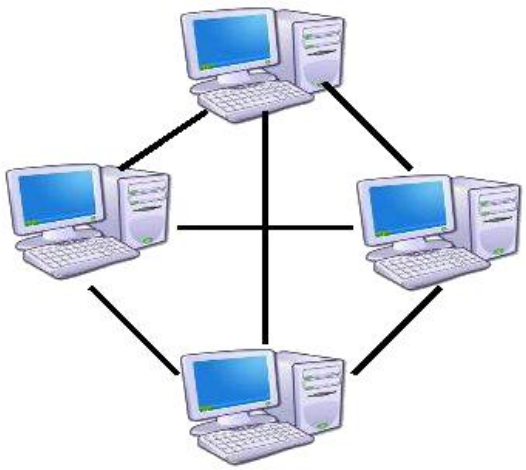


Fig. 1: Arquitectura P2P

Frente a este tipo de redes, se encuentran las que se basan en una arquitectura cliente-servidor, en la que los clientes solicitan recursos a uno o varios servidores. En este tipo de arquitectura, según aumenta el número de usuarios en la red, la tasa de transferencia disminuye. Esto ocurre porque los recursos de los que dispone el servidor se consumen debido al intenso tráfico que se genera.

En las redes P2P, cada nodo provee al resto de los recursos que dispone, obteniendo de esta forma un mayor rendimiento en las conexiones y en las transferencias. En la figura 2 puede verse un esquema de la arquitectura cliente-servidor.

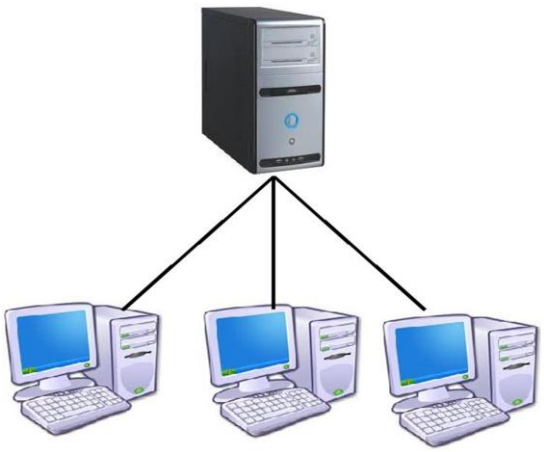


Fig. 2: Arquitectura cliente-servidor

Los aspectos fundamentales que caracterizan a una red P2P son los que se muestran a continuación.

- **Escalabilidad**. El alcance de las redes P2P es mundial, con un gran número de usuarios potenciales. Cuantos más usuarios estén conectados a la red, más recursos hay en el sistema y mayor rendimiento se obtiene.

- **Descentralización**. Por definición, estas redes son descentralizadas y los usuarios se comportan, simultáneamente, como clientes y servidores lo que hace que ningún nodo sea imprescindible para el correcto funcionamiento de la red.

- **Robustez**. La naturaleza distribuida y descentralizada de las redes P2P incrementa la robustez, ya que los peers no necesitan hacer peticiones a ningún servidor centralizado que pueda fallar.

- **Seguridad**. Es la característica menos implementada. Los objetivos de una red P2P segura serían identificar y evitar los nodos maliciosos, evitar el contenido infectado, evitar el espionaje en las comunicaciones entre nodos, protección de los recursos de la red, etc. Para ello existen mecanismos de cifrado de clave, comunicaciones seguras, gestión de derechos de autor, firmas digitales y demás métodos de seguridad.

- **Anonimato**. Es deseable que queden en el anonimato el autor de un contenido, el editor, el lector, el servidor que lo alberga y la petición para encontrarlo.

2.1.1. Clasificación de las redes P2P según su arquitectura

El modelo en el que se basa la arquitectura [2] de una red P2P puede ser centralizado, totalmente descentralizado o semicentralizado.

En el modelo centralizado, todas las solicitudes se realizan a través de un único servidor central, que sirve de punto de enlace entre los nodos de la red. Este servidor mantiene una base de datos en la que almacena la información de los archivos que tiene cada nodo, actualizándola cada vez que un cliente se conecta o desconecta. Cada solicitud de búsqueda es

comparada en el servidor con la información de la base de datos, y se contesta con las correspondencias encontradas. Una vez que el cliente tiene las correspondencias, se conecta directamente con cada peer y accede al recurso solicitado.

Este modelo se caracteriza por poseer una administración dinámica, una disposición más permanente de los recursos y un elevado rendimiento en su localización. Sin embargo, está muy limitado en lo referente a privacidad de usuarios, escalabilidad y robustez, ya que tiene puntos únicos de fallo y costos enormes en el mantenimiento y el consumo de ancho de banda.

Algunos ejemplos de redes que se basan en este modelo son Napster y Audiogalaxy. La figura 3 muestras una representación de redes P2P centralizadas.

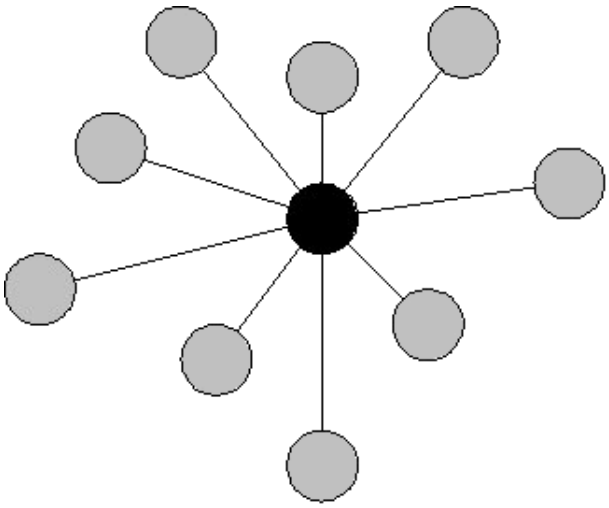


Fig. 3: Modelo centralizado

El modelo totalmente descentralizado o puro, figura 4, se caracteriza por no existir un servidor central, cada nodo actúa como cliente y como servidor. Cada peer trata de mantener conexiones con otros pares,

mandando y recibiendo peticiones y mensajes de control que facilitan el descubrimiento y encaminamiento. En otras palabras, todas las comunicaciones son directamente de usuario a usuario.

Las redes que se ajustan a este modelo son las más versátiles y robustas al no depender de un servidor central. No obstante, el elevado tiempo y la sobrecarga de ancho de banda que suponen las búsquedas de información son las principales desventajas de esta arquitectura.

Algunos ejemplos de redes que se basan en este modelo son Kademlia, Gnutella, Ares Galaxy y Freenet.

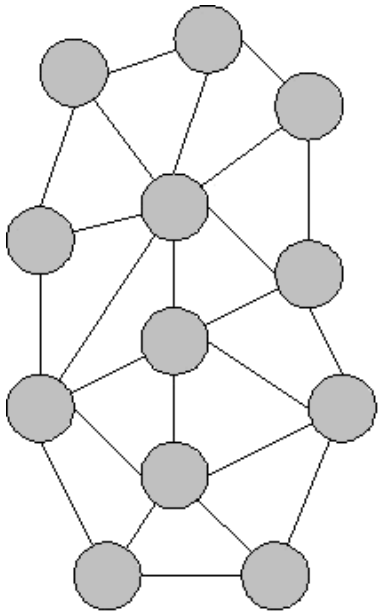


Fig. 4: Modelo descentralizado

El modelo semicentralizado o mixto es el más extendido entre las arquitecturas de redes P2P. En este modelo existe un servidor central especial, que ayuda a encaminar el tráfico y a administrar los recursos de la red sin conocer la identidad de cada nodo y sin almacenar información, por lo que no comparte ningún recurso. Puede darse en el caso en el que exista

más de un servidor que gestione los recursos compartidos. Cada nodo mantiene un número limitado de conexiones abiertas a un servidor y cada servidor está conectado con el resto de servidores de la red.

Este modelo se caracteriza por presentar un alto grado de escalabilidad, al reducir el número de nodos implicados en el proceso de encaminamiento, y disminuir el volumen de tráfico entre estos. En caso de que todos los servidores caigan, puede seguir funcionando la red ya que los nodos han establecido conexiones entre ellos.

Algunos ejemplos de redes que se basan en este modelo son eDonkey y BitTorrent.

En la figura 5, los nodos más oscuros hacen de servidores y se encargaran de gestionar los recursos de la red.

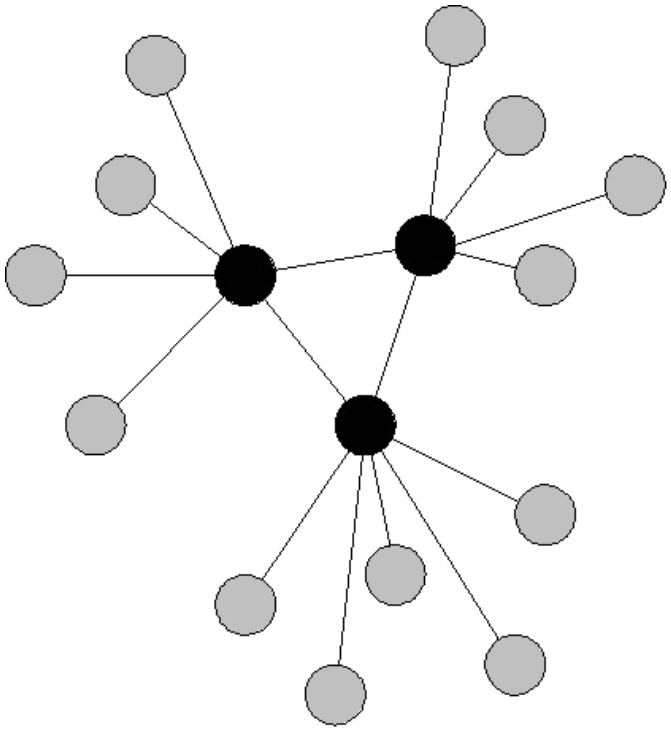


Fig. 5: Modelo semicentralizado

2.1.2. Clasificación de las redes P2P según su estructura

Las redes P2P se pueden clasificar [2] por tener o no una estructura determinada en base a como se enlazan unos nodos a otros.

Las redes P2P *estructuradas* son aquellas en las que los recursos están situados en nodos precisos. Cada nodo cuenta con su propia *tabla de hash*[1] y permiten que cada usuario sea responsable de una parte específica del contenido de la red. Estas redes utilizan una *función de hash*[2] para asignar valores a cada contenido y a cada usuario.

Algunos ejemplos de este tipo de redes son CAN y Chord.

En este tipo de redes se asegura la localización de un recurso siempre y cuando este se encuentre en la red, ya que proporcionan un mecanismo de búsqueda determinista. Además, los mensajes de búsqueda son encaminados de forma eficiente, es decir, un recurso es encontrado en O(log(n)) saltos, siendo n el tamaño de la red. El problema reside en que el usuario debe conocer exactamente el nombre del archivo para poder aplicar la función de hash y obtener un resultado adecuado.

Otro punto crítico se da debido al hecho de que cada vez que un usuario se conecta o desconecta de la red, se inicia un proceso de sincronización para actualizar las tablas de hash de los nodos vecinos. Esto crea una gran cantidad de mensajes de mantenimiento que en un caso extremo podrían dar lugar al colapso de la red.

En las redes P2P *no estructuradas*, la localización de recursos no está determinada en nodos concretos, a cada nodo se le asignan enlaces arbitrariamente, que irá actualizando. Utilizan un mecanismo de búsqueda no determinista que no garantiza que se vaya a encontrar el recurso aunque esté en la red. Las redes Gnutella son no estructuradas.

[1] Estructura de datos que asocia claves con valores.
[2] Algoritmo empleado para la generación de claves que representan valores de forma casi inequívoca.

Existen distintos mecanismos de búsqueda dentro de las redes no estructuradas pero el más conocido es la inundación o *flooding*. Cuando un nodo recibe un mensaje de solicitud de búsqueda de un recurso, si no conoce el recurso, reenvía el mensaje a todos los nodos con los que tiene una conexión. Este método tiene como inconveniente que introduce mucho tráfico en la red.

2.1.3. Redes overlay estructuradas

2.1.3.1. Chord

Chord [3] es un tipo de red peer-to-peer descentralizada y estructurada que utiliza una *función de hash* para asignar identificadores únicos de *m* bits a los nodos y datos que conforman la red.

El identificador de nodo se obtiene aplicando la función de hash sobre la dirección IP del nodo, mientras que el identificador de un dato, a partir de ahora *key*, se genera aplicando la *función de hash* sobre el recurso.

La topología de una red Chord es básicamente un anillo de 2^m elementos en el cual los nodos están ordenados según sus identificadores, en orden creciente siguiendo el sentido de las agujas del reloj. Cada *key K* se asigna al primer nodo cuyo identificador sea igual o superior a *K* en el anillo.

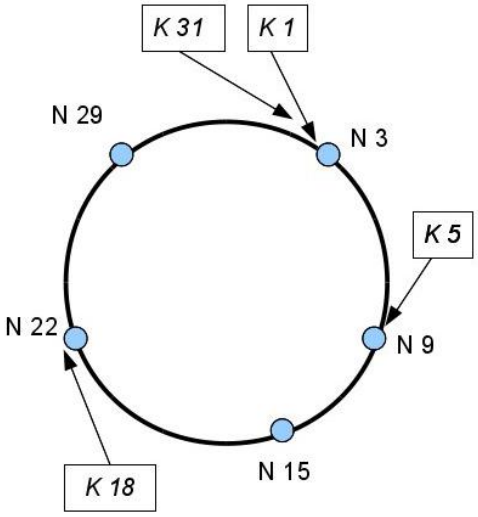


Fig. 6: Anillo de Chord

La figura 6 muestra un ejemplo de distribución de peers (Nx) y datos (Kx). Los nodos se ordenan en un anillo en orden creciente y las keys se asignan a los nodos correspondientes.

Chord está diseñado para ser flexible y permitir que los nodos entren y salgan de la red produciendo el menor impacto posible en la estructura. Cuando un nodo se une a la red, se le asignan todas las *keys* de las cuales era responsable su sucesor, que no están entre el nuevo nodo y el sucesor. Cuando un nodo abandona la red, todas las *keys* de las que era responsable se le asignan a su sucesor.

En la figura 6, si el N22 abandona la red, N29 se hará responsable de K18.

Recorrer el anillo en busca de un elemento puede ser muy ineficiente utilizando sólo el puntero al sucesor de cada nodo. Para mejorar la búsqueda, cada nodo tiene una *finger table* que contiene punteros a nodos estratégicamente elegidos. El algoritmo para elegir los punteros o *fingers* es muy sencillo, el $finger^{ith}$ apunta al nodo que es igual o sucede al identificador 2^{i-1}. La tabla 1 muestra el cálculo de la *finger table* correspondiente al nodo 3

de la figura 7.

sucesor(3+2^{1-1}) = sucesor(4) = N5

sucesor(3+2^{2-1}) = sucesor(5) = N6

sucesor(3+2^{3-1}) = sucesor(7) = N9

sucesor(3+2^{4-1}) = sucesor(11) = N15

sucesor(3+2^{5-1}) = sucesor(19) = N22

Tabla 1: Cálculo de finger table de N3

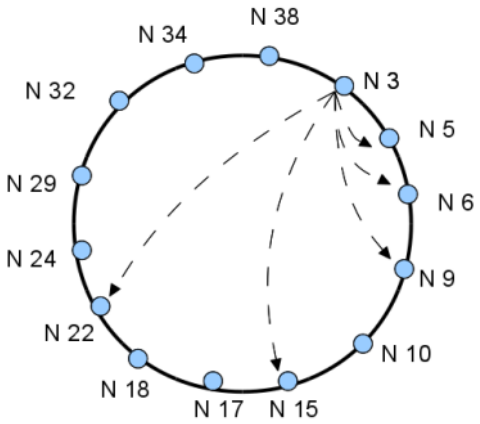


Fig. 7: Finger table de N3

Si se da el caso de que un nodo no tiene suficiente información en su *finger table* para encontrar un *key*, este enviará la solicitud al nodo más cercano a la información buscada con identificador menor.

En el caso de la figura 7 y tabla 1, si N3 busca K27 almacenada en N29, N3 enviará la query a N22.

El número medio de peers por los que debe pasar una query viene dado por la fórmula $\frac{1}{2}\log_2 N$, siendo N el número total de peers en la red [4].

Cuando un nodo nuevo entra en la red [5] se producen una serie de acciones para conservar la estructura de la red y asegurar el correcto funcionamiento. El nuevo nodo tiene que saber cuales son su predecesor y su sucesor. El sucesor tiene que fijar como predecesor el nuevo nodo y el predecesor tiene que poner el nuevo nodo como su sucesor.

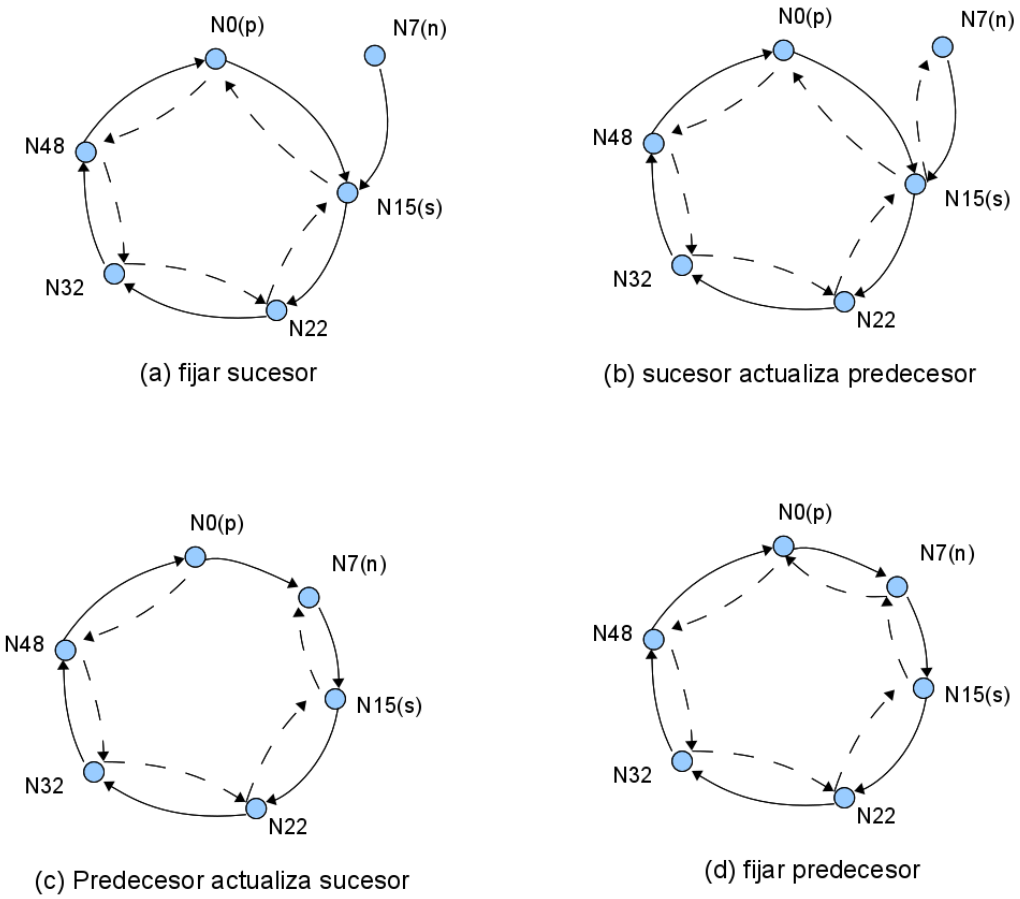


Fig. 8: Operación de *join*

La figura 8 muestra los pasos de una operación de *join*. En (a) el nodo que se quiere incorporar a la red, N7, detecta que su sucesor va a ser N15 y guarda esa información. A continuación, en (b), informa a N15 de que su predecesor cambia y N15 actualiza su puntero. En (c) el sucesor de N0 se actualiza, siendo ahora el nodo que se incorpora a la red y en (d) finaliza el proceso de *join* actualizando el nuevo nodo su puntero del predecesor.

2.1.3.2. Kademlia

Kademlia [6] es un protocolo de comunicaciones para redes descentralizadas y estructuradas peer-to-peer. Cada nodo tiene un nodeID, esto es, un número binario único que lo identifica dentro de la red y se genera aplicando una *función de hash* a la dirección IP. Para obtener el *key* de un dato se procede aplicando esa misma función sobre el dato.

Cuando un nodo necesita una información, la busca en los nodos que considera más cercanos al *key* y cuando necesita almacenarla, lo hace en el nodo que considera más cercano al *key* asociado.

Un nodo Kademlia almacena información de sus contactos en *buckets* que contienen un máximo de k contactos. Los *buckets* están organizados según la distancia entre el nodo y sus contactos.

Por ejemplo, para el *bucket* j, donde 0<=j>k,

$$2^j <= \text{distancia (nodo, contacto)} < 2^{j+1}$$

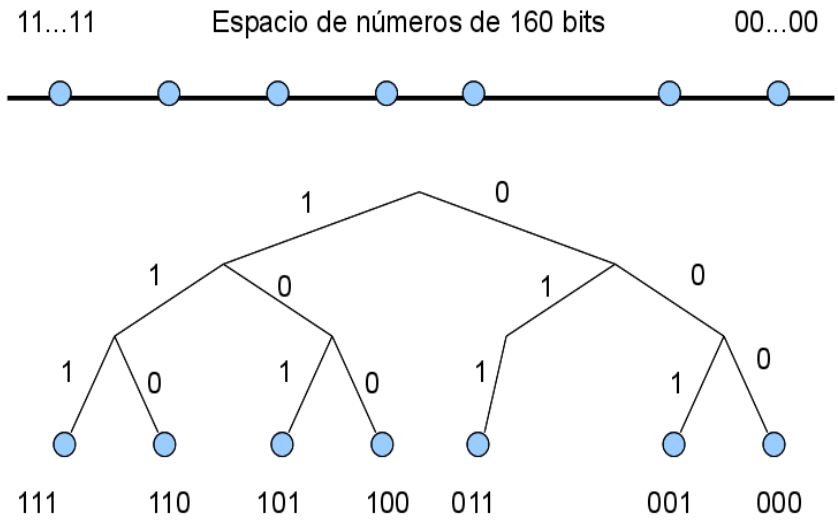


Fig. 9: Árbol binario Kademlia

La figura 9 muestra la estructura del árbol binario Kademlia. El algoritmo utilizado por Kademlia se basa en el cálculo de la distancia entre dos identificadores (de nodo o información) para asignar los pares <key, value> a un nodo concreto. Esta operación se lleva a cabo mediante el uso de la operación XOR:

$$\text{distancia}\ (x,\ y) = x \otimes y$$

Para asegurar la existencia de los pares <key,value>, estos debes ser publicados periódicamente por los nodos. Siempre que un nodo recibe algún tipo de mensaje de otro, este actualiza el *bucket* correspondiente. Si el contacto ya está registrado en el nodo, este se mueve al final del *bucket*. Si el contacto no existe como tal y el *bucket* que le corresponde no está lleno, se inserta al final.

Si el *bucket* está lleno, el nodo envía mensajes ping al contacto que están en la cabecera del *bucket*. Si este nodo no contesta al ping, es eliminado del *bucket* y se introduce el nuevo contacto en la cola del mismo. Si no, el nuevo contacto es ignorado en cuanto a actualizaciones del *bucket* se refiere.

Búsqueda de un nodo

La búsqueda empieza seleccionando los *alpha*[3] contactos más cercanos al *key* sobre el que se realiza la búsqueda. Estos contactos están en sus *k-buckets*. Si en el *bucket* seleccionado hay menos de *alpha* contactos, se seleccionan de otros *buckets*. Los primeros *alpha* contactos más cercanos al *key* objetivo se almacenan en una lista de nodos preseleccionados.

El nodo empieza a enviar de forma paralela y asíncrona mensajes de búsqueda de datos y búsqueda de nodos a los alpha contactos. Cuando un

[3]Parámetro que caracteriza la cantidad de búsquedas simultáneas en una red Kademlia.

nodo recibe esta solicitud, devuelve los datos (nodeID, IP y puerto) de los k contactos más cercanos al *key*.

Si alguno de los alpha contactos falla, se le elimina de la lista. El nodo rellena la lista con los contactos de las respuestas que ha recibido. Se vuelven a seleccionar alpha contactos de la lista y se envían otra vez los mensajes.

La única condición que deben cumplir estos nodos, es que no se haya contactado antes con ellos. Este procedimiento continúa hasta que ningún nodo de los recibidos esté más cerca que el más cercano de los que ya están en la lista.

Incorporación de un nodo a la red

Para que un nodo se una a la red debe realizar una serie de operaciones. Tiene que conocer la dirección IP y el puerto de algún nodo que ya se encuentre en la red Kademlia. Una vez que tiene estos datos, se genera un nodeID si no tiene ya uno, inserta este valor en el bucket apropiado del nodo conocido, realiza una búsqueda de si mismo y actualiza los contactos de sus *buckets* con la información obtenida.

2.1.4. Redes overlay jerárquicas

Una red overlay jerárquica [7] es aquella en la que los nodos participantes se organizan en grupos, los cuales están a su vez comunicados mediante otra red overlay. Este tipo de redes tratan de mejorar las características de las redes peer-to-peer tradicionales como son el tamaño de la tabla de rutas, el retardo y el rendimiento del enrutado.

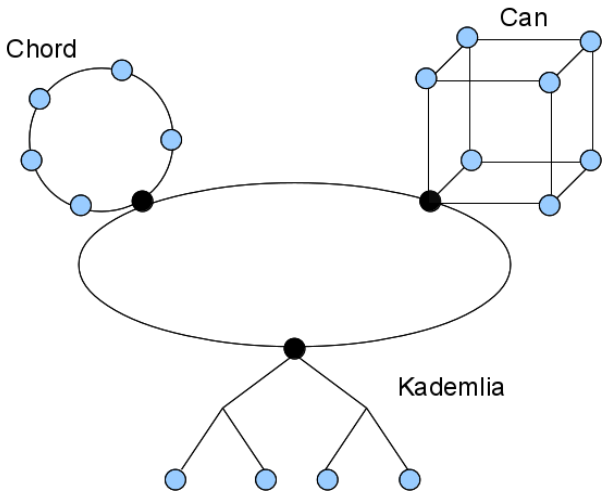


Fig. 10: Red overlay peer-to-peer jerárquica

La figura 10 ilustra un ejemplo de una red peer-to-peer jerárquica [8] en la que la red de interconexión utiliza el protocolo Chord y conecta redes Kademlia, Can y Chord.

Cada nodo del nivel de interconexión crea un grupo y pasa a ser el super-peer de ese grupo. Éste actúa como pasarela entre los peers de su grupo y los de otros grupos.

Existen dos tipos de búsqueda: las búsquedas en su propio dominio y las búsquedas en otro dominio. En el primer caso, la búsqueda se realiza como si de una red plana se tratara. En el segundo caso, el super-peer se encarga de que el mensaje llegue al peer correspondiente encaminándolo a través de la red de interconexión hacia el dominio correspondiente.

Los dominios pueden tener más de un super-peer para hacer la red más robusta y pueda continuar activa si un super-peer deja de funcionar. Los nodos seleccionados para ser super-peers serán los más potentes y estables ya que tienen que soportar todo el tráfico que les llega de otras subredes. En caso de que sólo exista un super-peer para cada grupo, cuando este falla, otro tiene que reemplazarlo y realizar todas sus funciones.

Un nodo que se une a una red jerárquica tiene que saber de antemano a cuál de las subredes debe hacerlo. Si aún no hay conectado ningún nodo de su dominio, este nodo se convierte en el super-peer y crea su propia subred.

2.1.5. Aplicaciones de las redes P2P

En la actualidad muchos servicios y aplicaciones requieren una gran capacidad de almacenamiento y de ancho de banda, unos recursos caros. Estas aplicaciones [9], pueden hacer uso de las redes P2P. Algunos ejemplos a destacar son:

- **Búsqueda e intercambio de ficheros**. Probablemente sea la aplicación más extendida de este tipo de redes. Los propósitos son muy diferentes estando entre ellos el uso personal y los fines académicos. Algunos ejemplos son eDonkey2000 y BitTorrent.

- **Televisión P2P**. Sistemas para la difusión y transmisión de contenidos audiovisuales a través de Internet. Los nodos se conectan a otros nodos para recibir los *streams*[4] de video y audio en lugar de conectarse a un servidor central. SopCast es un ejemplo de este tipo de aplicación.

- **Sistemas de telefonía por Internet**: Sistemas que permiten la transmisión en tiempo real de señales de voz por la red, como Skype.

- **Sistemas de ficheros distribuidos**. Sistemas de datos en los que la información se almacena en nodos de la red y se permite el acceso de otros pares a la misma. Un ejemplo de este tipo de aplicación es Freenet.

[4] Flujo de información entre un origen y un destino.

2.2. Churn en las redes Peer-to-Peer

El número medio de nodos que abandonan la red en un determinado intervalo de tiempo se denomina churn.

Es un factor muy importante en las redes Peer-to-Peer ya que la continua entrada y salida de nodos hace que las tablas de rutas sean poco consistentes y no reflejen la topología real de la overlay. Esto ocurre especialmente cuando las salidas de peers no han sido señalizadas de forma adecuada debido a fallos o perdidas de conectividad.

Según estudios [10] sobre el comportamiento del churn en redes DHT se comprueba como el tiempo entre llegadas y salidas de nodos se asemeja a una distribución binomial negativa.

2.2.1. Actualización de las tablas de rutas

La actualización de las tablas de rutas es un aspecto muy importante para paliar los efectos del churn y existes diversos estudios al respecto [11]. Básicamente hay dos formas de realizar esta actualización.

La primera es la recuperación reactiva y consiste en actualizar cualquier cambio de la tabla de rutas de un peer si se detecta alguna entrada errónea. Esa información es enviada a los nodos vecinos tan pronto como sea posible.

La otra opción es la recuperación periódica. Ésta actualiza la información periódicamente y los cambios en la tabla de rutas son enviados a los vecinos sólo si expira un periodo de tiempo prefijado .

2.2.2. Réplica de información

Cuando un nodo falla debido al churn, los datos de los que este era responsable se vuelven inaccesibles y se deben poder recuperar de alguna forma. Un algoritmo de duplicación [12] soluciona este problema almacenando copias de la información en determinados nodos.

En Chord, cada nodo guarda información en su *finger table* sobre sus vecinos más cercanos en el anillo en sentido de las agujas del reloj, es decir, sus sucesores.

Las réplicas se almacenan en los r sucesores del nodo responsable de esa información. Para mantener estas réplicas actualizadas y que el efecto del churn no haga que se pierda información existen dos mecanismos: uno de mantenimiento local y otro global.

El mantenimiento local consiste en que cada nodo envía a sus r sucesores una lista con la información de la que es responsable y los sucesores se encargan de actualizar sus bases de datos.

Cuando el protocolo usado es el de mantenimiento global, cada nodo comprueba de forma periódica su base de datos para saber si hay datos que no debería almacenar. Para ello busca al nodo responsable de cada información que tiene guardada y comprueba si él mismo está en su lista de sucesores. Si es así, mantiene el dato, si no, lo elimina.

Capítulo III

3. Diseño e implementación de una red overlay jerárquica Chord en el simulador PeerfactSim.KOM

3.1. Simulador PeerfactSim.KOM

Existen distintos simuladores disponibles [13] en Internet que ofrecen la posibilidad de trabajar con redes Peer-to-peer. Estos simuladores varían en cuanto a su complejidad, implementación, facilidad de uso, requerimientos del sistema…etc. Finalmente se decide utilizar PeerfactSim.KOM debido a que utiliza el lenguaje Java, es fácil de programar, es extensible y se tenía experiencia con él en otros proyectos.

PeerfactSim.KOM nace de la necesidad de un nuevo simulador adecuado para redes peer-to-peer complejas. El objetivo es diseñar un marco de simulación que sea escalable, orientado a objetos y ligero para una simulación eficiente y precisa de la red P2P. Es importante que sea lo más parecido posible al mundo real.

Debido a la complejidad de estas redes, el modelado de sistemas P2P debe cumplir ciertos requisitos. Se necesitan mecanismos eficientes para modelar el comportamiento de los usuarios y los detalles de los protocolos. Por otra parte, hay que tener muy en cuenta las características de la red física subyacente ya que puede tener aspectos cruciales a la hora de diseñar la red P2P.

El comportamiento de los usuarios también es un punto crítico al que hay que prestar la suficiente atención. Éste puede ser bastante impredecible, desde estar en línea constantemente compartiendo muchos archivos hasta permanecer activo sólo cuando es necesario para obtener los archivos deseados y no compartir los propios con el resto de usuarios de la red.

En el anexo III puede encontrarse un estudio más detallado sobre la arquitectura del simulador así como diferentes características a tener en cuenta a la hora de implementar una overlay en PeerfactSim.KOM.

3.2. Implementación de partida

Para llevar a cabo este proyecto se parte de la implementación de una red Peer-to-peer chord de un solo nivel realizada por Yue Sheng. Esta implementación forma parte del simulador PeerfactSim.KOM.

El código de la red Chord está dividido en tres paquetes clasificando así las clases en operaciones (join, lookup, store…), mensajes (notificaciones, estabilización, ping…) y aspectos generales de la red Chord como pueden ser el nodo, el identificador de nodo o la finger table. Estos tres paquetes están incluidos en el paquete *de.tud.kom.p2psim.impl.overlay.dht.chord* que podemos ver en la figura 42 en el anexo III.

A continuación se muestran las clases de cada paquete con sus respectivos métodos y herencias:

Aspector generales:

En este paquete se encuentran las clases necesarias para crear cada uno de los elementos que forman la red. El elementeo principal es el ChordNode. Para que se cree un nodo, la clase ChordNodeFactory ejecuta el método createComponent(), que este a su vez llamara al constructor de

ChordNode. Los nodos tienen una serie de elementos que están representados en forma de atributos y le dan distintas funciones.

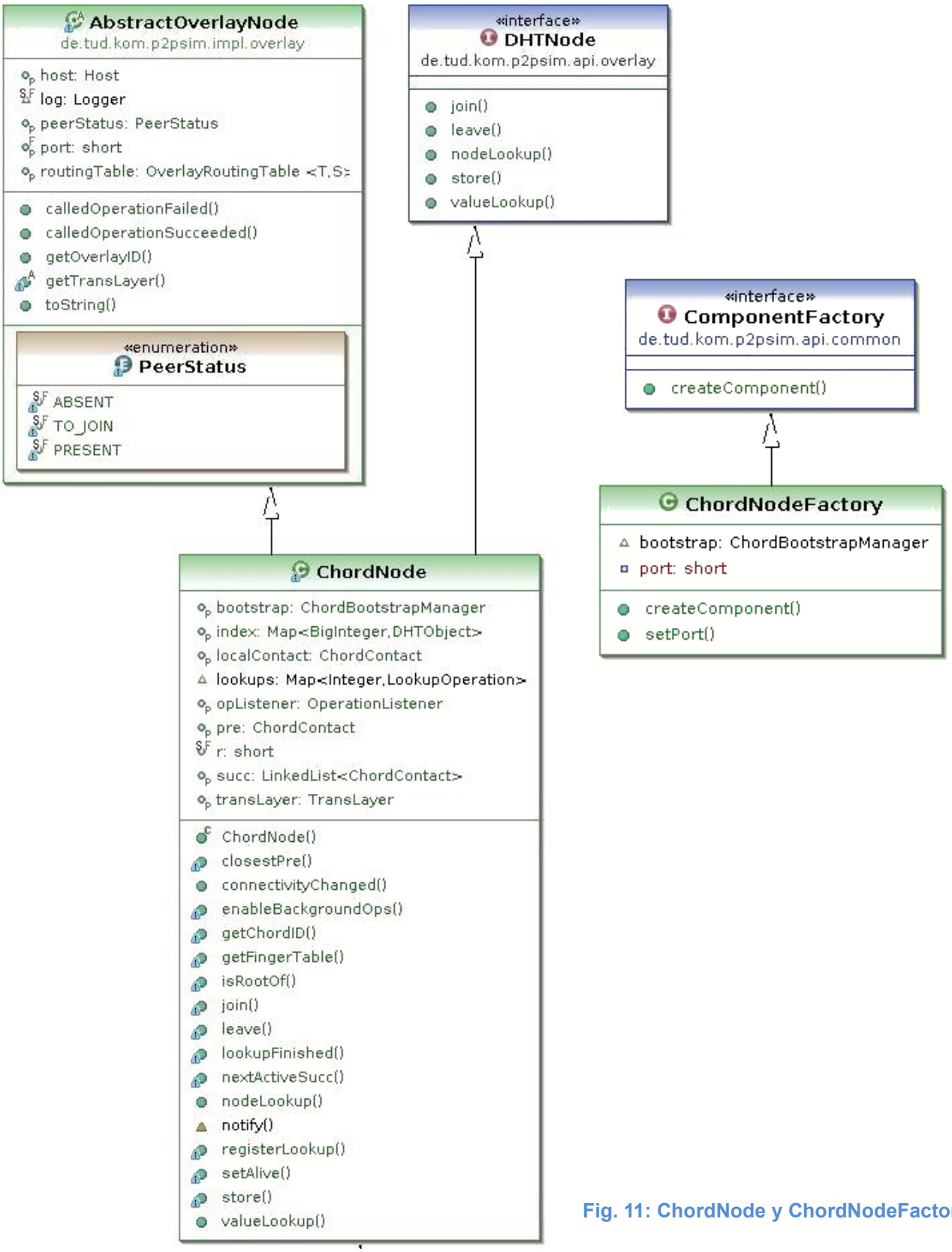


Fig. 11: ChordNode y ChordNodeFactory

El identificador de cada nodo y cada dato viene definido en las clases ChordID y ChordKey. Las clases DHTObjectParser, OverlayKeyParser y ChordIDGenerator son las encargadas de generar los valores que tomarán estos identificadores.

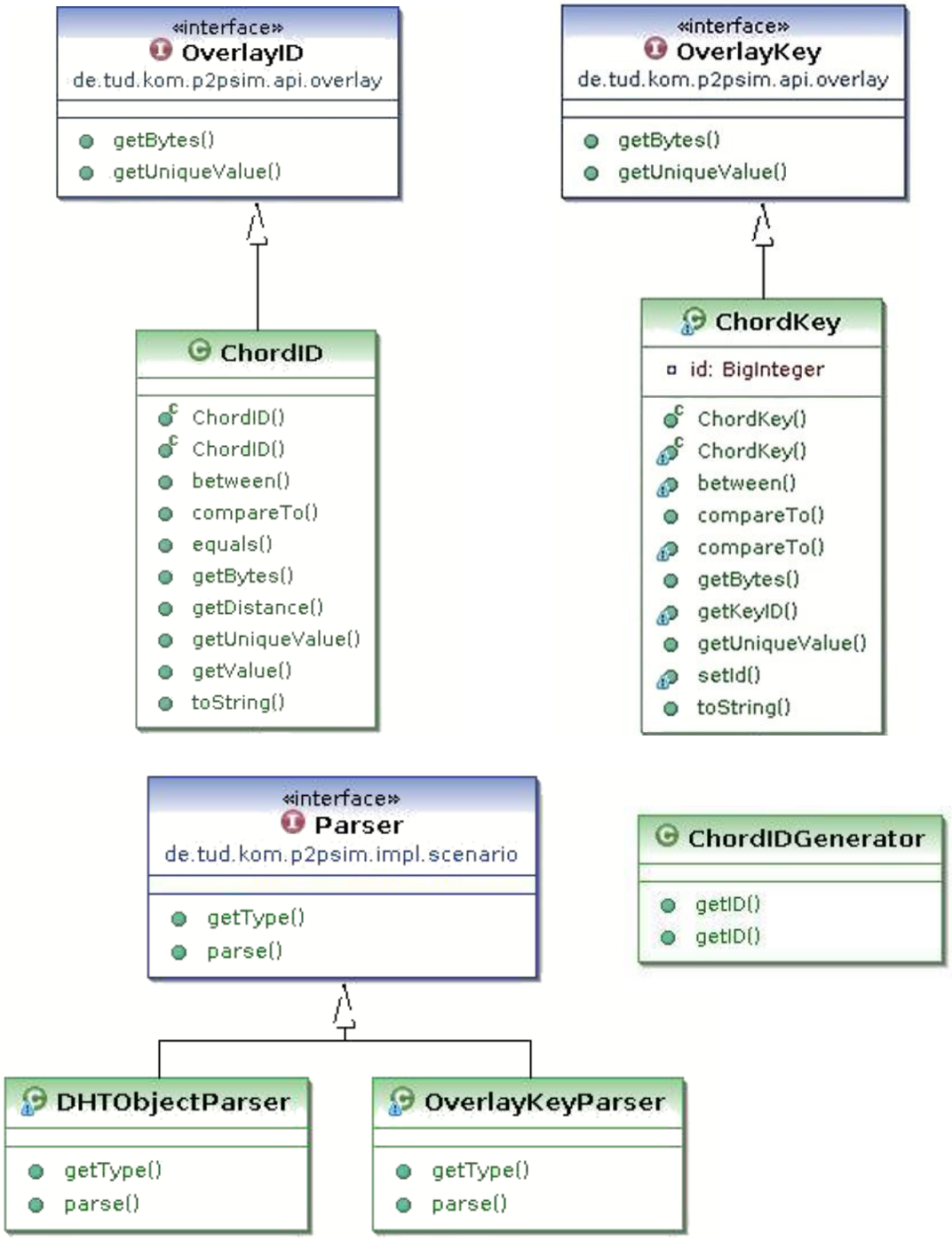


Fig. 12: ChordID, ChordKey, DHTObjectParser, KeyParser y ChordIDGenerator

La tabla en la que guardan la información sobre otros peers que están en la red se crea mediante la clase FingerTable. Cada uno de los elementos guardados en la finger table es un ChordContact. Estos contactos almacenan el identificador de un nodo, su dirección para ponerse en contacto con él y si está activo o inactivo.

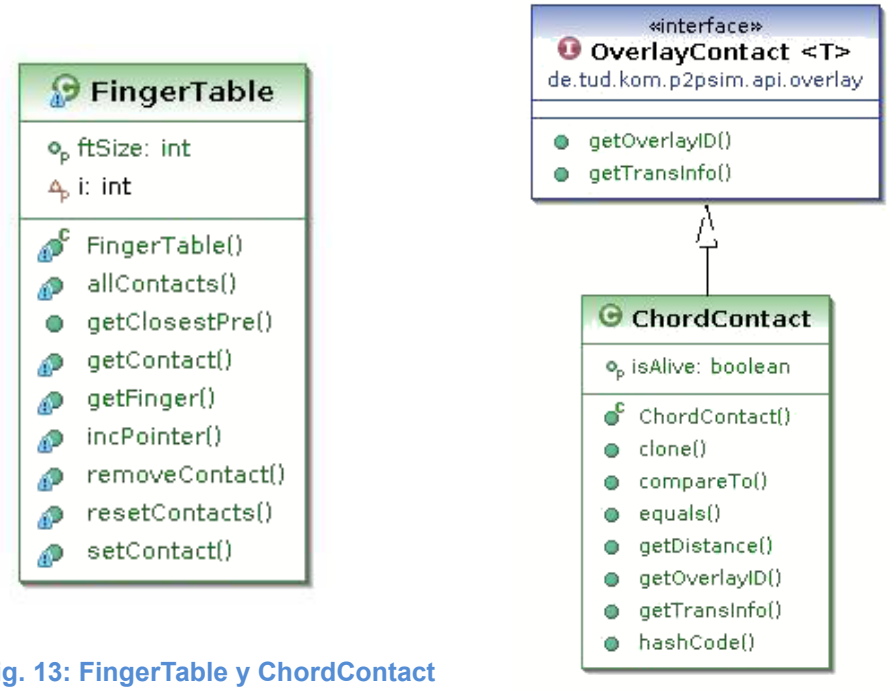


Fig. 13: FingerTable y ChordContact

ChordBootstrapManager mantiene una lista de 5 nodos activos en la red. Esta clase se utiliza cuando un nodo nuevo quiere acceder a la red ya que de ella obtiene el primer contacto con el que se relaciona.

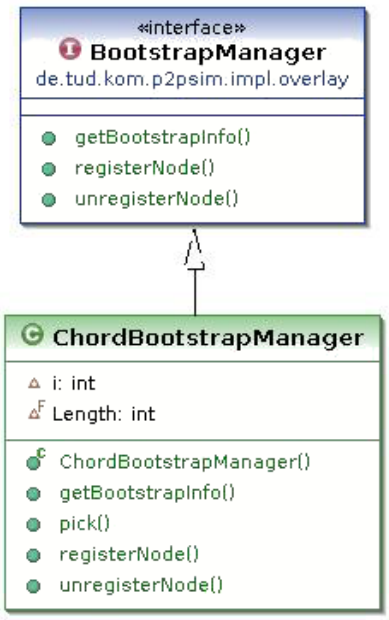


Fig. 14: ChordBootstrapManager

Por último, todos los nodos tiene dos clases encargadas de gestionar los mensajes y las operaciones periódicas que ejecutan, estas son ChordMessageHandler y OperationListener.

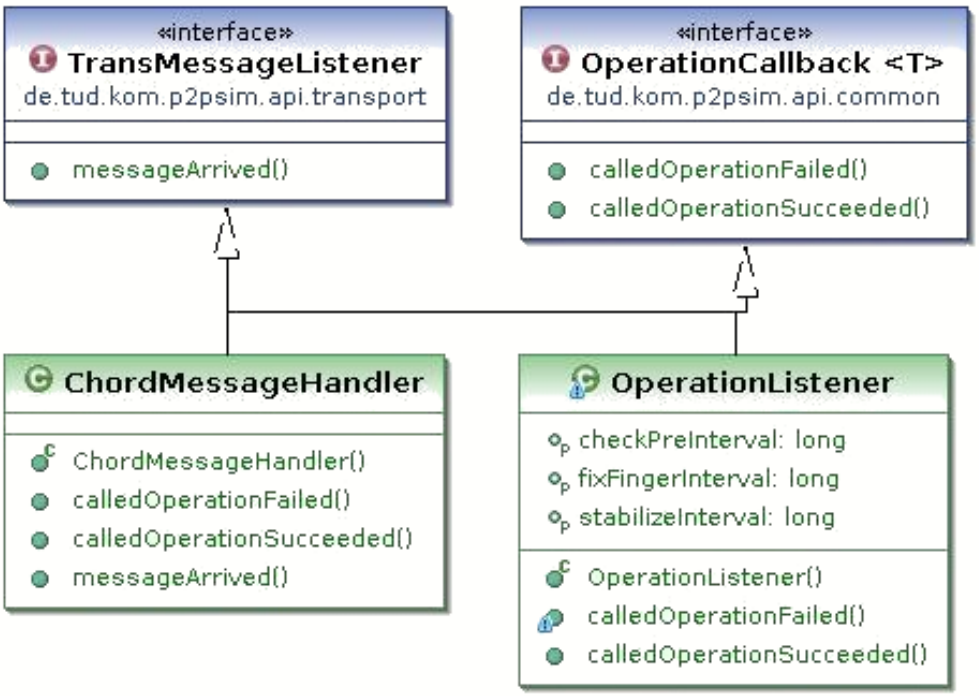


Fig. 15: ChordMessageHandler y OperationListener

Mensajes:

Los nodos necesitan intercambiarse información tando a la hora de realizar operaciones como en la búsqueda de archivos de datos. Todo esto va encapsulado en distintos mensajes cada uno de los cuales tiene una función. En la siguiente página, la figura 16 muestra un esquema con todos ellos. El propio nombre de las clases define la función del mensaje al que se refiere.

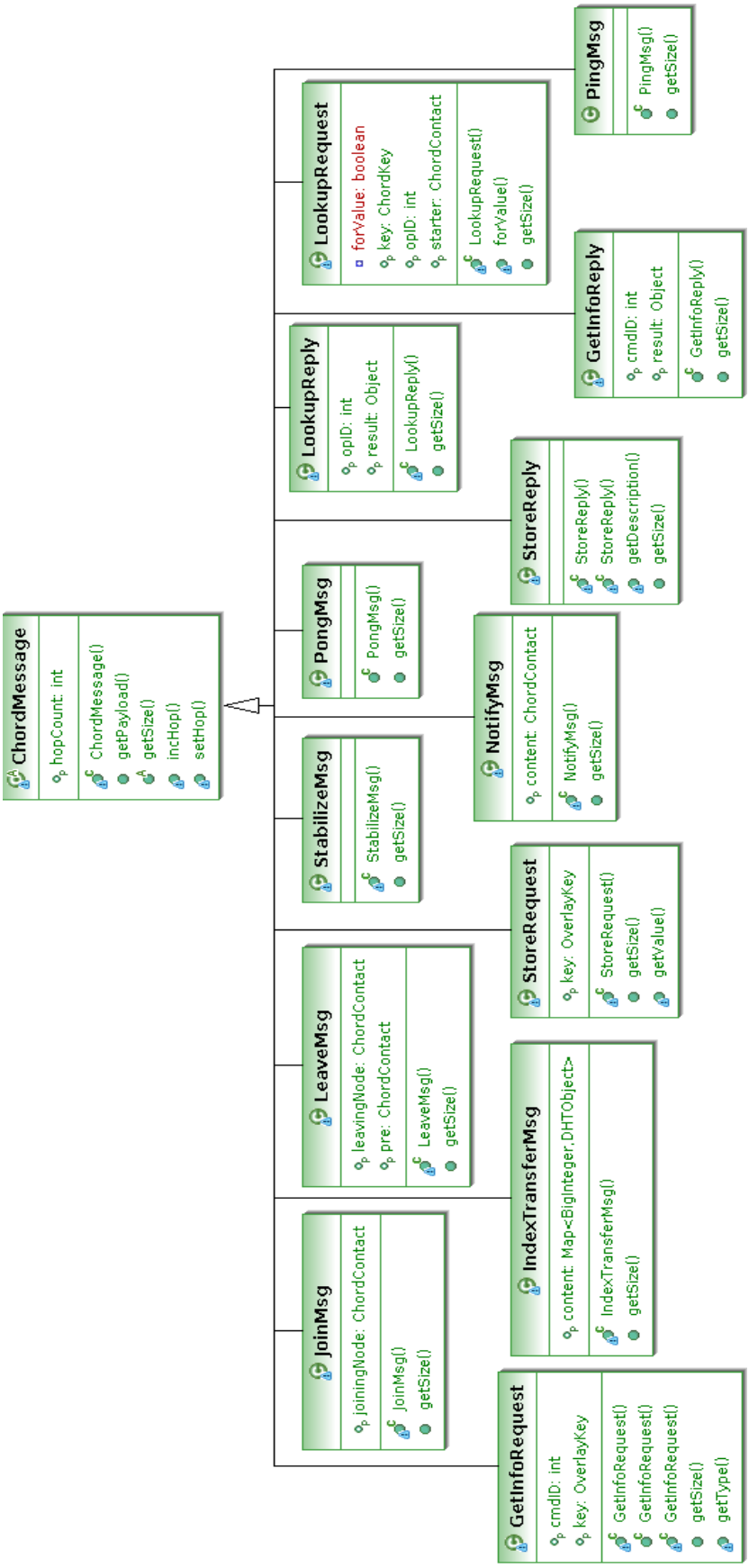


Fig. 16: Esquema de mensajes en Chord

Operaciones:

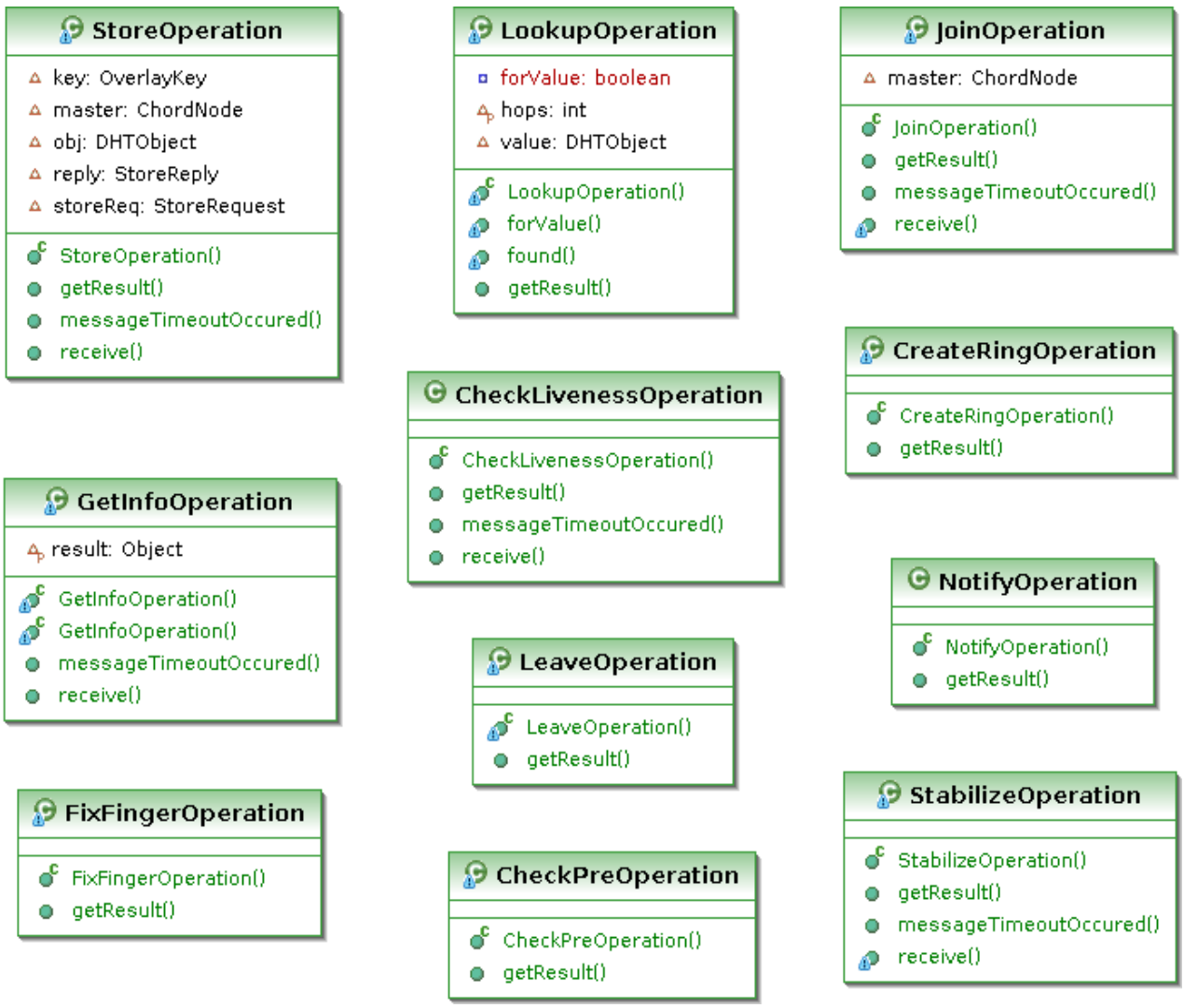


Fig. 17: Operaciones en Chord

FixFinger, CheckLiveness, CheckPre y Stabiliza se ejecutan para mantener actualizada la información que alamacena cada peer sobre algunos de los nodos que componen la red para mantener la topología.

Join, Store, Leave y Lookup son las 4 operaciones básicas que se llevan a cabo en la red. CreateRing se encarga de crear la red a la que posteriormente se van a unir los peers y el resto de operaciones son utilizadas para completar a las 9 de las que ya se ha hablado.

3.3. Requisitos de una red overlay jerárquica

Aunque los nodos pueden estar organizados en varios niveles, en este proyecto se trabaja únicamente con redes de dos niveles. Se tiene un nivel superior formado por super-peers y un nivel inferior formado por nodos normales. A cada uno de los grupos del nivel inferior se les llama dominios y los dominios están conectados entre si mediante la red de interconexión.

Esta implementación se basa en un diseño [14] que permite tener cualquier tipo de red estructurada en cada uno de los dominios.

Existen una serie de características que diferencian a la red jerárquica de la red plana y que hay que tener en cuenta a la hora de realizar la implementación:

- **Peers separados en grupos**: Todos los nodos que componen la red están divididos en grupos. Estos grupos están unidos mediante la red de interconexión formada por los super-peers. Cada uno de los grupos es totalmente independiente del resto, dos grupos sólo pueden comunicarse entre sí a través de sus super-peers. En este caso, se trabaja sólo con el protocolo Chord pero podría darse el caso en el que incluso los protocolos de enrutamiento de los distintos dominios fueran diferentes.

- **Identificadores jerárquicos:** Es necesario un identificador que sea capaz de definir tanto el nodo o información a la que se refiere, como el dominio en el que está ubicado. Se divide el identificador en dos partes**:**

Fig. 18: Identificador jerárquico

El prefix-ID indica el dominio y es el que se tiene en cuenta a la hora de encaminar mensajes por la red de interconexión cuando el destinatario se encuentra en otro dominio. El suffix-ID identifica al nodo o información concreta. Una vez que una búsqueda se encuentra ya en el dominio del destinatario, el enrutamiento se hace mediante este campo del identificador.

- **Búsqueda jerárquica:** Existen dos tipos de búsqueda: las búsquedas de un valor en su propio dominio (estando en el dominio A, buscar x@A) y las búsquedas de un valor en otro dominio (estando en el dominio A, buscar x@B). En el primer caso, la búsqueda se realiza como si de una red plana se tratara. En el segundo caso, cuando un nodo detecta que el valor buscado no forma parte de su dominio, debe enviar el mensaje al super-peer responsable de su subred. El super-peer realiza la búsqueda del super-peer responsable del dominio destino en la red de interconexión y le envía el mensaje. Este super-peer realizará la búsqueda normal en su dominio como si de una red plana se tratara.

3.4. Modificaciones en la red plana

Antes de añadir las nuevas funcionalidades para convertir la red plana en jerárquica se han realizado una serie de modificaciones para obtener un mejor funcionamiento.

3.4.1. Réplicas

La implementación de la que se parte carece de un mecanismo de duplicación de información para asegurar su continuidad en la red una vez que el nodo responsable de ella la abandona de forma imprevista.

Para suplir este posible problema, se diseña una función de réplica consistente en almacenar en los r sucesores del nodo responsable de la información, copias de esa información. Esta acción se lleva a cabo cuando un nodo decide que quiere compartir un dato e inicia una operación de store.

A continuación se crea una operación nueva, llamada HReplicationOperation, según la cual, durante la simulación, cuando cambia el predecesor del peer o alguna entrada de la finger table, el nodo comprueba de que pares <key, value> es responsable y almacena réplicas en sus r sucesores, se utilizará r=5. De esta forma la probabilidad de que se pierda una información se reduce de manera considerable.

Esta nueva operación genera un tráfico extra que antes no existía y que es necesario estudiar. Se barajó la opción de que HReplicationOperation se lanzara de forma periódica cada 10 segundos de igual manera que las operaciones de mantenimiento de la red. Tras realizar distintas pruebas, se comprobó que la solución con la que mejores resultados se obtenía es con la finalmente implementada.

La figura 19 muestra el esquema de la clase HReplicationOperation. *NumReplicas* es el atributo que indica el número de réplicas con el que se trabaja y este se puede variar según las necesidades de la red.

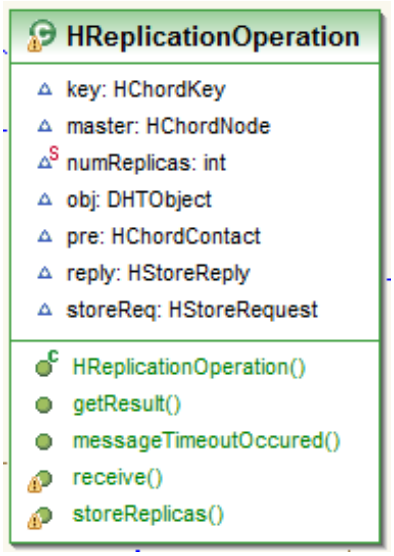


Fig. 19: HReplicationOperation

3.4.2. Reintento en caso de fallo

Lo ideal en una red de ordenadores es que todas las operaciones que se lleven a cabo entre ellos finalicen correctamente después del primer intento. Debido a diversos problemas que pueden surgir en la red, como la congestión, la desactualización de las tablas de rutas, etc..., esto es prácticamente imposible. En la red estudiada en este proyecto, las operaciones que no concluyen de forma satisfactoria, vuelven a ser ejecutadas pasado un tiempo suficiente para que el mecanismo de estabilización haya concluido y las tablas de rutas estén correctamente actualizadas.

3.4.3. Churn

El número de peers en una red P2P varía continuamente con la llegada y salida de nodos en todo momento. Para introducir churn en la red y que las simulaciones sean lo más reales posible, se crea la operación leave según la cual el nodo que la ejecuta se desconecta y abandona la red sin informar a ningún otro nodo. Para que todo siga funcionando correctamente, la red realiza una serie de operaciones de estabilización y actualización de tablas de rutas.

En la red estudiada, el churn se genera de forma aleatoria antes de iniciar la simulación. Está incluido en el archivo de operaciones .dat del que se habla en el capítulo 4.1.2 y en el Anexo II de la presente memoria.

Según aumenta el tamaño de la red en las simulaciones, mayor será el número de nodos que abandone la red y a continuación recupere la conexión. La manera más acertada de modelar [10] estos dos parámetros es utilizando una distribución binomial negativa que variará la media según el número de nodos que forman la red.

El churn se activa una vez que la red está totalmente formada pero antes de empezar la recogida de datos con el fin de que alcance un estado lo más estable posible y estos sean fiables para su estudio.

3.5. Adaptación a red jerárquica

Para conseguir una red jerárquica con dos niveles y completar todas las funcionalidades requeridas es necesario modificar algunas clases y crear otras. Todo ello se ha hecho modificando lo menos posible la estructura original del protocolo.

En todas las clases ha sido necesario introducir unos métodos y atributos que sean capaces de trabajar con los nuevos identificadores jerárquicos. En determinadas clases se han realizado una serie de modificaciones que se muestran a continuación.

3.5.1. Elementos de la red

3.5.1.1. Identificador de nodo e información

Las clases responsables de estos identificadores son HChordID y HChordKey respectivamente. Como se ha explicado en el apartado 3.3 relacionado con los requisitos de una red jerárquica, los nodos y datos necesitan que una parte de su identificador defina el dominio al que pertenecen.

Para ello se ha creado un campo llamado *netID* de características idénticas al que identifica el nodo dentro de su dominio. El identificador de un nodo o dato queda definido por dos campos de la siguiente forma:

- **netID:** define el dominio al que pertenece el nodo o en el que está almacenado el dato.

- **Id:** identifica al nodo o al dato dentro del dominio.

Si un nodo o dato esta definido por “a@x”, siendo “a” el nombre del nodo o dato y “x” la red a la que pertenece, el *netID* se obtiene al aplicar la función de hash sobre “x” y el *id* se obtiene al aplicar la función de hash sobre “a”. Todos los nodos y datos del mismo dominio compartirán el mismo *netID*.

La posición que ocupa un peer dentro de un dominio depende del valor del *id*. La posición que ocupa un super-peer en la red de interconexión depende del valor del netID.

Los métodos que existían en la red plana para trabajar con los identificadores han tenido que ser modificados, sin cambiar su funcionalidad, para darles la capacidad de trabajar con el nuevo *netID*.

Las figuras 20 y 21 muestran las clases en las que se definen estos dos identificadores. Se puede ver como heredan de las clases de la red plana y ocmo tiene como atributo nuevo la parte de identificador netID. Se observa también como en el caso de HChordID tenemos el método *betweenInter()* y *getNetIDUniqueValue().* Estos dos métodos trabajan sobrela red de interconexión.

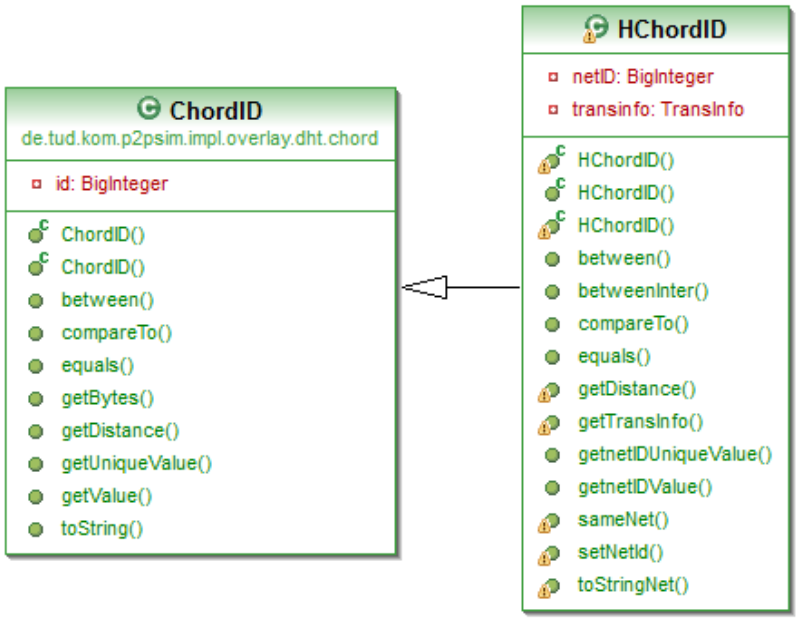


Fig. 20: HChordID

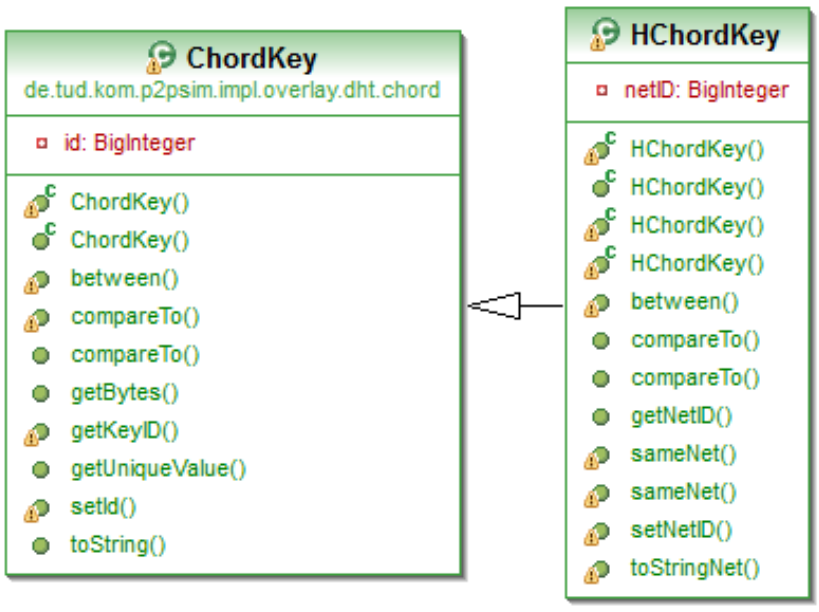


Fig. 21: HChordKey

3.5.1.2. Peers y super- peers

La clase HChordNode crea los peers de cada dominio. La única variación que presenta es la creación de un método que devuelve el super-peer responsable de la red a la que pertenece el nodo. Su utilidad radica en que cuando se realiza una query cuyo destinatario no se encuentra en el dominio propio, esta se envía inmediatamente al super-peer en un único salto.

La clase HChordSuperNode implementa los nodos que trabajan como super-peers, es decir, los que, además de realizar las funciones de un peer normal en el dominio al que pertenecen, añaden una serie de funcionalidades que les permiten interactuar en la red de interconexión con los nodos que la conforman. Debido a esto, hereda de la clase HChordNode todos los atributos y métodos y a parte se crean nuevos:

- Bootstrap de interconexión o superBootstrap. Contiene hasta 5 nodos activos pertenecientes a la red de interconexión que le permiten unirse a ella. Funciona de la misma manera que el bootstrap original y es utilizado a la hora de crear el anillo de interconexión.

- Finger table de interconexión. Se utiliza cuando un super-peer recibe una query con un netID distinto al suyo. En este caso accederá a la finger table de interconexión para obtener el siguiente salto de la red de interconexión que le acerque más al super-peer de la red destino.

- InterconexionSucc e interconexionPre. Almacenan los punteros a los nodos sucesores y al predecesor en la red de interconexión.

Ha sido necesario implementar los mismos métodos que se heredan, con el mismo procedimiento de trabajo, pero con capacidad para trabajar en la red de interconexión. Estos métodos duplicados así como los nuevos atributos pueden verse en la figura 22.

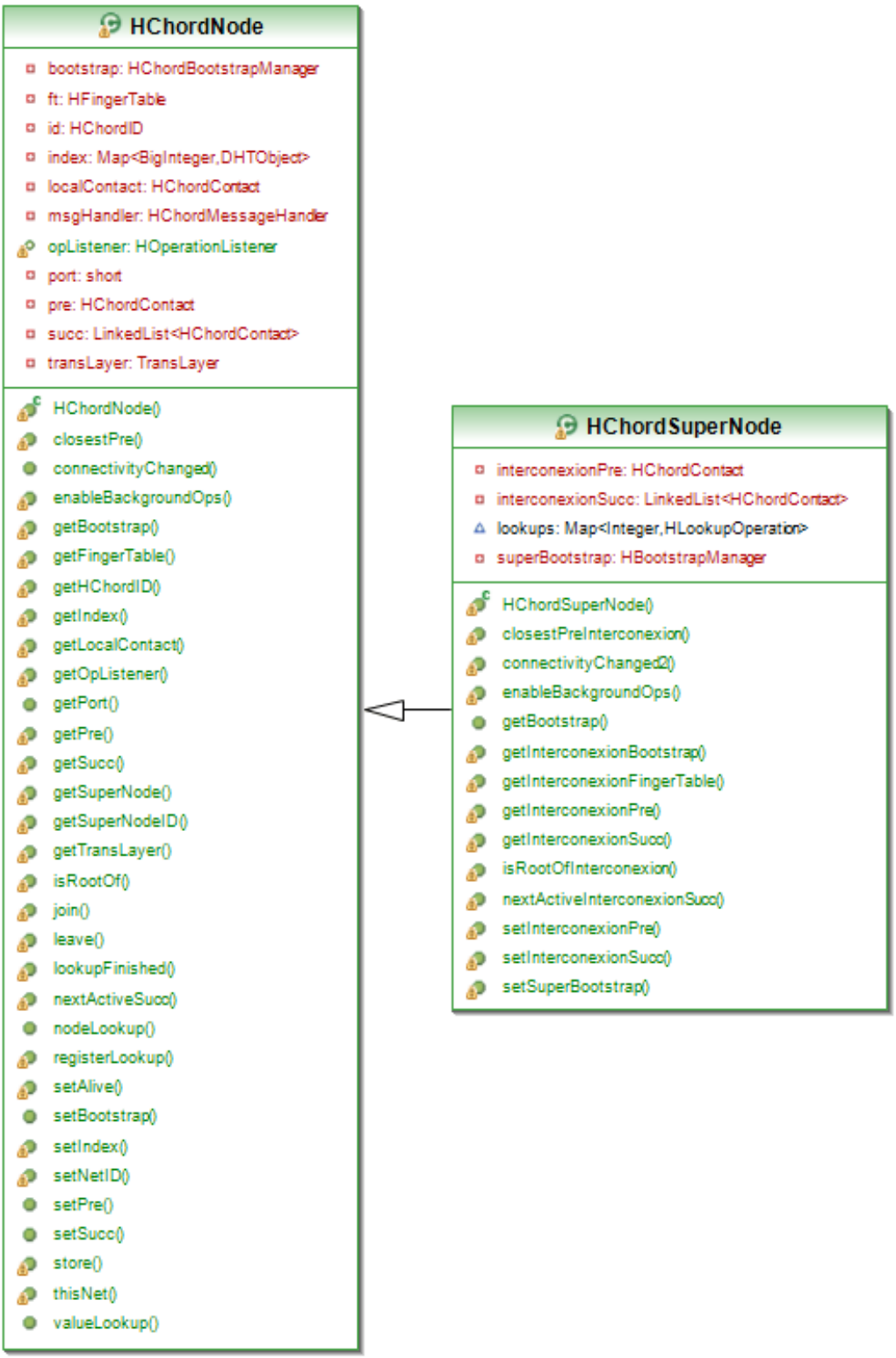


Fig. 22: HChordSuperNode

3.5.1.3. Administración de mensajes entrantes

HChordMessageHandler es un atributo que todos los nodos tienen asociado y se encarga de administrar todos los mensajes que le llegan. Cuando un nodo recibe un mensaje, según sea este, debe decidir como actuar.

Con la mayoría de tipos de mensajes el procedimiento no varía con respecto a la red plana. Sólo en los casos de mensajes de "join", "getInfoRequest" y "HLookupRequest", se realiza una única modificación que consiste en introducir una estructura de selección *if-else* para decidir si se trabaja sobre la red de interconexión o sobre alguno de los dominios.

La figura 23 muestra el diagrama de decisión de un peer cuando recibe una query y la figura 24 la estructura de la clase.

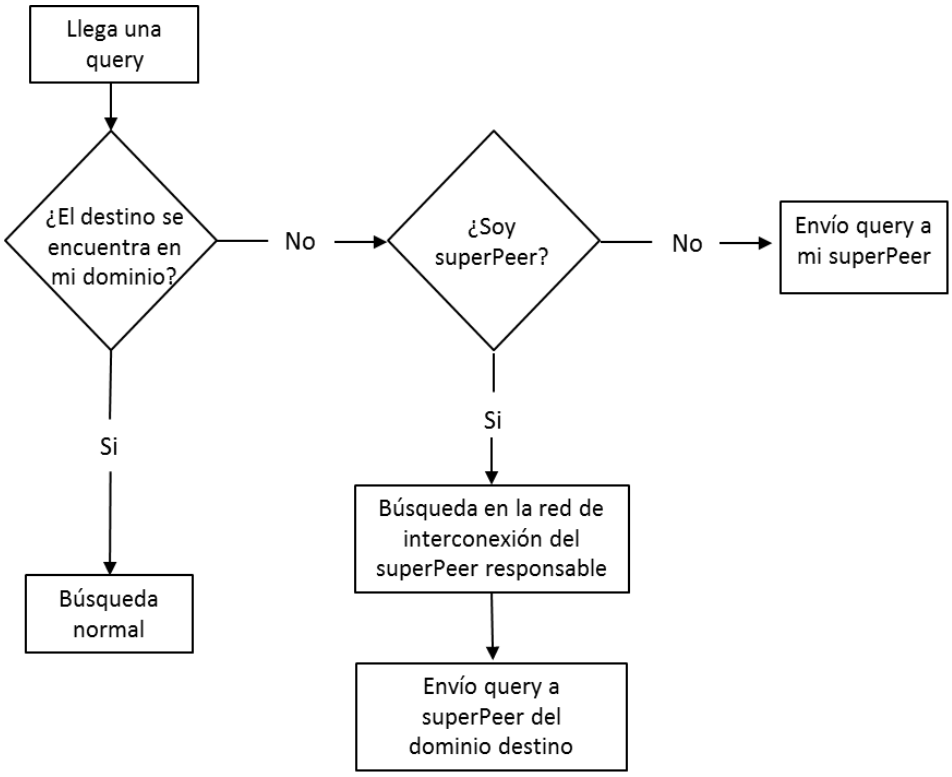


Fig. 23: Organigrama de decisión de un peer al recibir una query

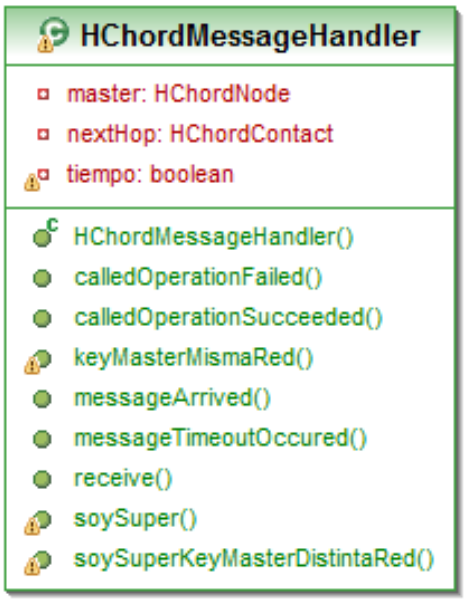


Fig. 24: HChordMessageHandler

3.5.1.4. *Bootstrap manager*

En esta clase ha sido necesario introducir un atributo que almacena el super-peer asociado a ese dominio. De esta forma, el peer que tiene asignado ese bootstrap tiene comunicación directa en un solo salto con su super-peer y se reduce de manera considerable el número de saltos necesarios para llegar a un peer de otro dominio. Este nuevo atributo está representado en la figura 25 como *superNode*.

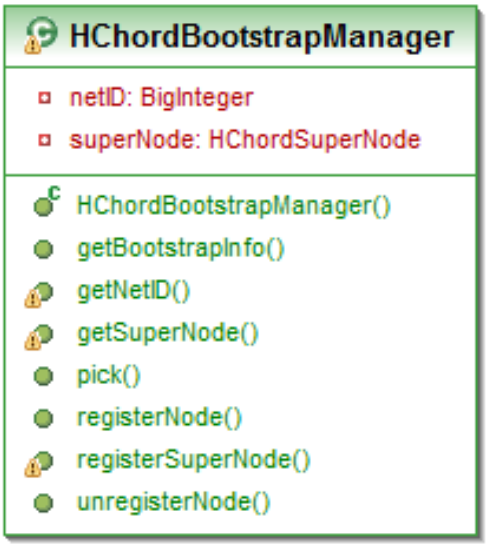


Fig. 25: HChordBootstrapManager

3.5.1.5. ChordNodeFactory

Esta clase entra en funcionamiento al principio de la simulación y es la encargada de crear los nodos. Hereda de HNodeFactory(), clase creada para que sea capaz de trabajar con nodos de distintos protocolos cuando se implemente la opción jerárquica en la que no sólo se trabaje con Chord. El método createComponent() comprueba si tiene que crear un super-peer o un peer y les asignar los bootstrap correspondientes. Si se trata de un super-peer, llama al método createSuperPeer() y este se encarga de asignarle el bootstrap del dominio y el correspondiente a la red de interconexión. La figura 26 muestra los componentes de esta clase.

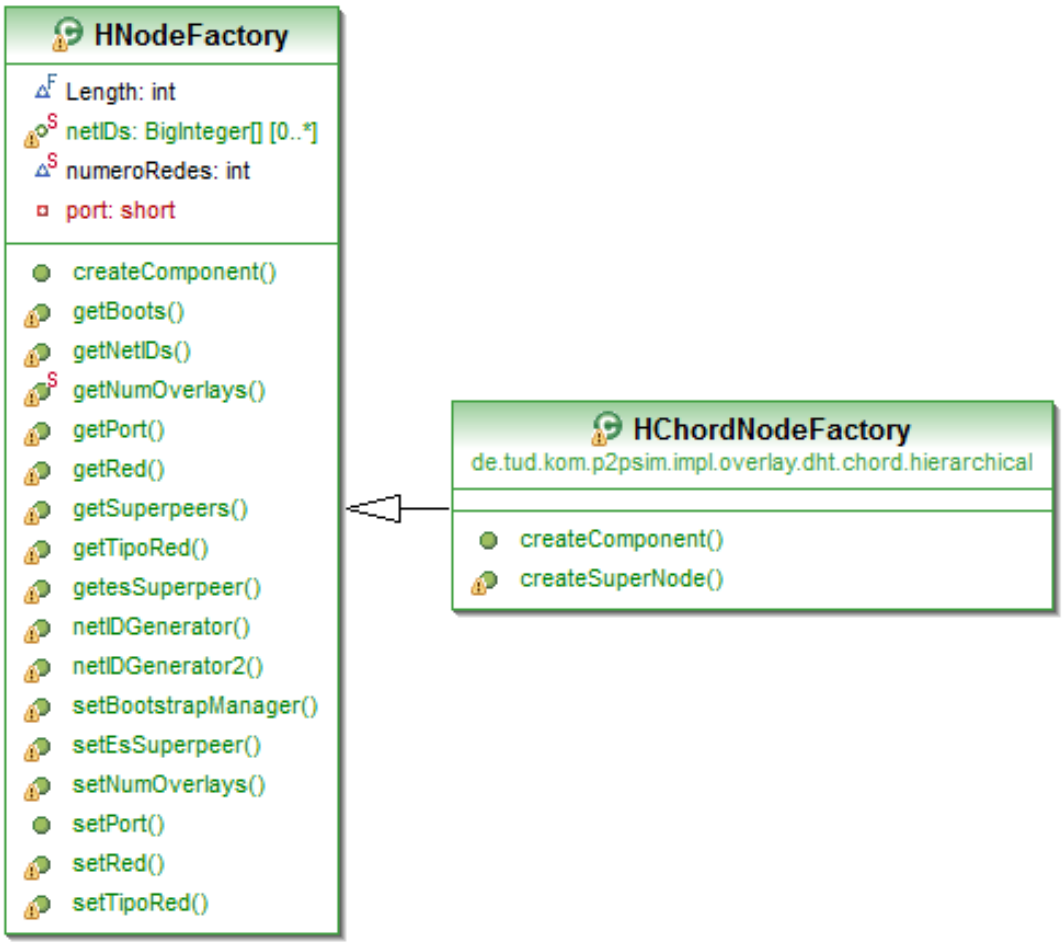


Fig. 26: HChordNodeFactory

3.5.2. Mensajes

Los mensajes que se envían por la red de interconexión y por los dominios son iguales. Están representados en la figura 16 del apartado 3.2. No hay más cambios que los necesarios para que sean capaces de trabajar con los nuevos identificadores de nodo.

3.5.3. Operaciones

3.5.3.1. Creación de la red o anillo de interconexión

La nueva clase HCreateInterconexionRingOperation() se encarga de crear la red de interconexión. El mecanismo es el mismo que el de la creación de los dominios de nivel inferior. Este método se llama una única vez a lo largo de la simulación ya que sólo el primer super-peer, es el que crea la red de interconexión.

En la figura 27 se puede ver como esta clase no tiene relación directa con los peers normales ya que la red de interconexión está únicamente formada por super-peers.

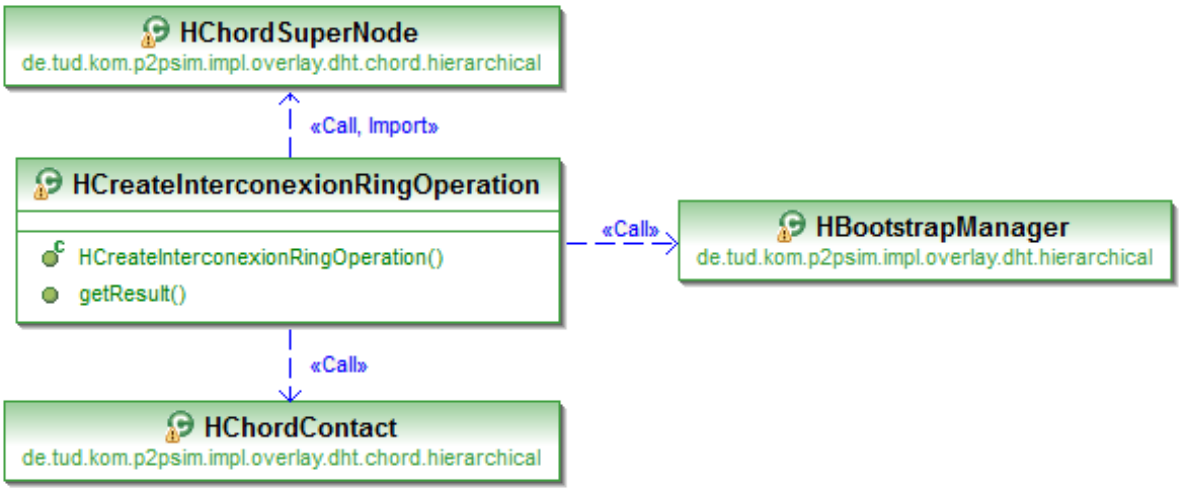


Fig. 27: HCreateInterconexionRingOperation

3.5.3.2. Join

Un super-peer tiene que ser accesible al mismo tiempo tanto por la red de interconexión como por el dominio al que pertenece, para ello, cuando se une a la red, tiene que hacerlo a ambas.

Primero comprueba si algún super-peer se ha unido con anterioridad y ha creado la red de interconexión. En caso de que no haya sido así, este nodo es el encargado de hacerlo creando una operación HCreateInterconexionRingOperation(). La figura 28 muestra los atributos y métodos de esta clase así como las relaciones correctas con otras clases.

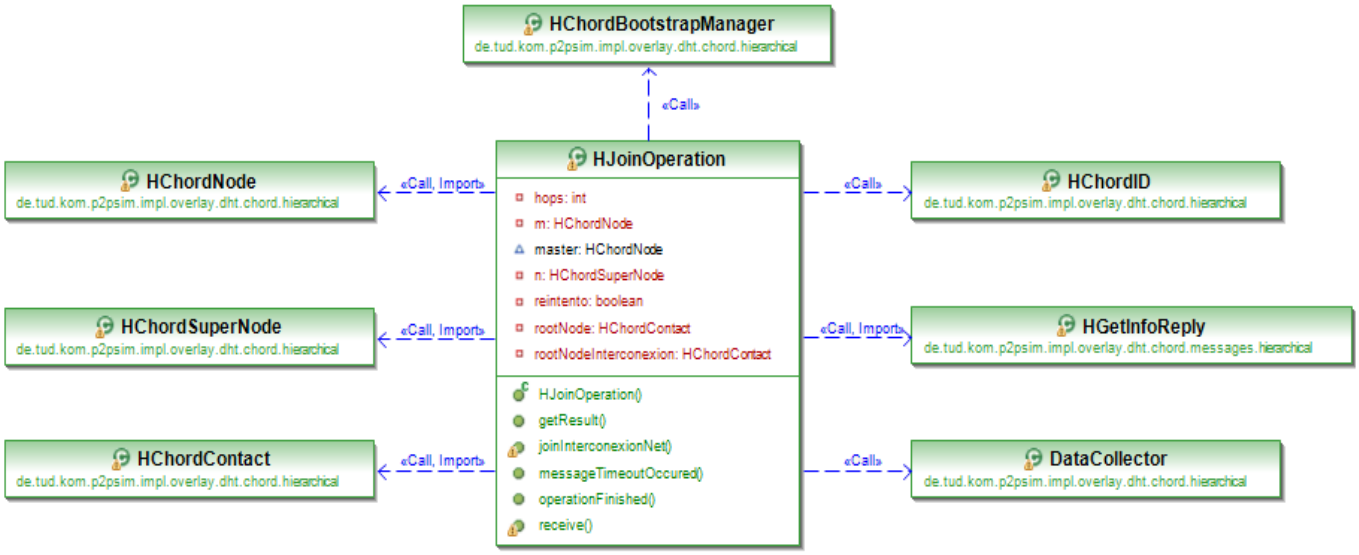


Fig. 28: HJoinOperation

Una vez que existe la red de interconexión, el super-peer se une a ella y crea la red del dominio inferior al que va a pertenecer. A continuación, se une a ella. Como puede verse, un super-peer pertenece a las dos redes.

Los dos procesos de join son iguales y se realizan tal y como se describe en el apartado 2.1.3.1 y se muestra en la figura 8 de este documento. La única diferencia radica en la parte de identificador de nodo

que se considera en cada uno de los casos para la ordenación de los peers en la red. Los super-peers se ordenan en la red de interconexión según sea el valor de su *netID* y al dominio de nivel inferior según el valor del *id*. En ambos casos lo hacen en orden creciente y siguiendo el sentido de las agujas del reloj.

En la figura 29 se muestra la situación de una red chord jerárquica cuando sólo existen super-peers. Como se puede ver, todos los super-peers forman parte de la red de interconexión pero cada uno de ellos tiene su propio dominio.

Suponiendo que SP1 haya sido el primero, habrá sido el responsable de crear la red de interconexión a la que se han unido posteriormente SP4, SP6 y SP9.

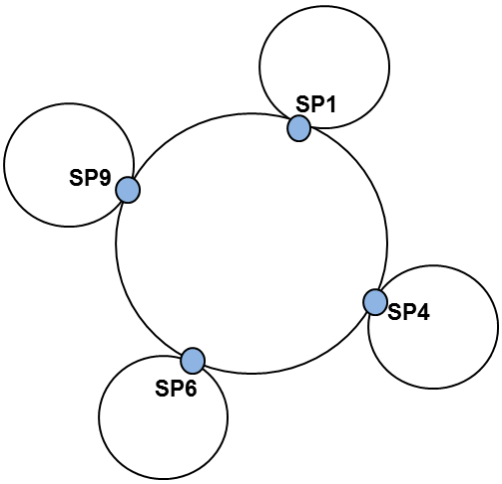


Fig. 29: Red chord jerárquica con solo super-peers

3.5.3.3. Leave

La operación HLeaveOperation es la utilizada para generar el churn. Ya que el hecho de que un peer abandone la red de forma ordenada y avisando a sus vecinos no conlleva ningún problema, cada vez que un nodo abandone la red lo hará de forma inesperada y sin enviar ningún tipo de

mensaje. En el caso de la red implementada, existe la condición de que un super-peer no pueda abandonar la red. Si alguno lo intentara, esta operación es la encargada de comprobar que eso es imposible.

3.5.3.4. Lookup

En una red jerárquica existen dos tipos de operaciones de búsqueda, las intra-dominio y las inter-dominio. En las primeras, el origen y el destino están en el mismo dominio, la operación se lleva a cabo como si de una red plana se tratara.

La operación inter-dominio se complica con respecto a la red plana ya que la búsqueda tiene que pasar por el dominio origen, la red de interconexión y el dominio destino. Se puede dividir en tres pasos.

El primer paso consiste en enviar el mensaje al super-peer del dominio. Cada nodo del dominio tiene un puntero a su super-peer por lo que este paso se realiza en un solo salto. En el caso de esta red, este puntero se mantiene durante toda la simulación ya que el super-peer no cambia y no se ve afectado por el churn. Este paso queda representado en la figura 30.

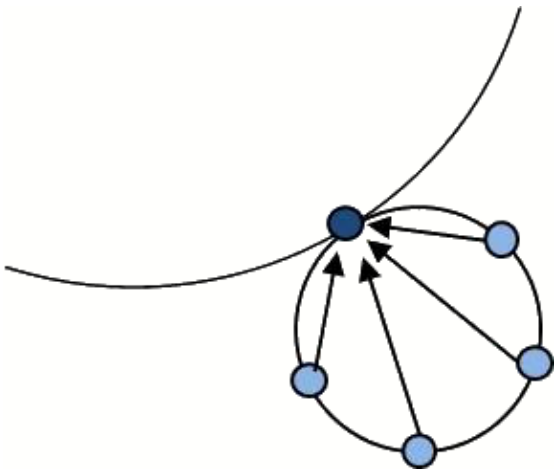


Fig. 30: Primer paso, puntero al super-peer

En el segundo paso el super-peer trabaja como pasarela entre los dominios y el anillo de interconexión. Cuando una query llega a un super-peer, encamina la búsqueda por la red de interconexión utilizando el netID del valor buscado hasta que encuentra un super-peer con el mismo netID. Este enrutamiento es igual que el utilizado en Chord y descrito en el apartado 2.1.3.1. La figura 31 ilustra un mensaje encaminado entre super-peers por la red de interconexión.

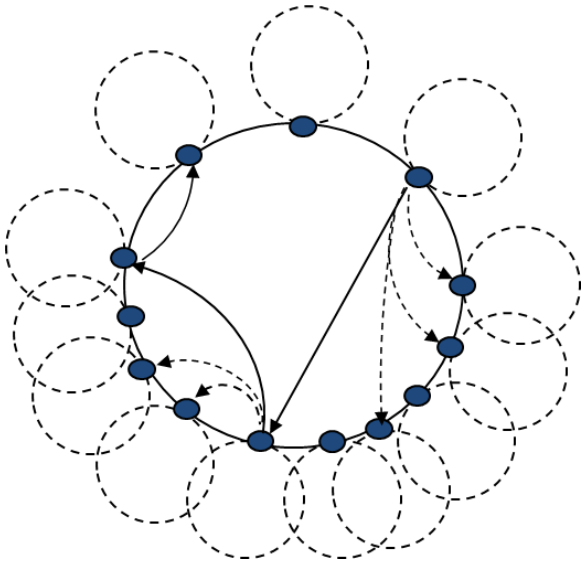


Fig. 31: Segundo paso, red de interconexión

El tercer paso se lleva a cabo cuando se ha encontrado al super-peer correcto en la red de interconexión, es decir, el super-peer responsable de dominio en el que se encuentra el nodo destino. El enrutamiento es igual que el utilizado en Chord y descrito en el apartado 2.1.3.1. La figura 32 muestra este último paso de la operación de lookup en una red jerárquica.

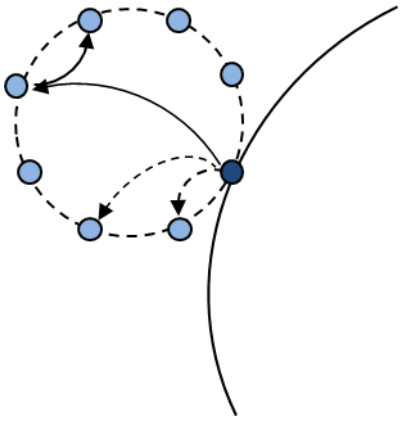


Fig. 32: Tercer paso, encaminamiento en el dominio destino

A continuación se muestran los atributos y los métodos de la clase.

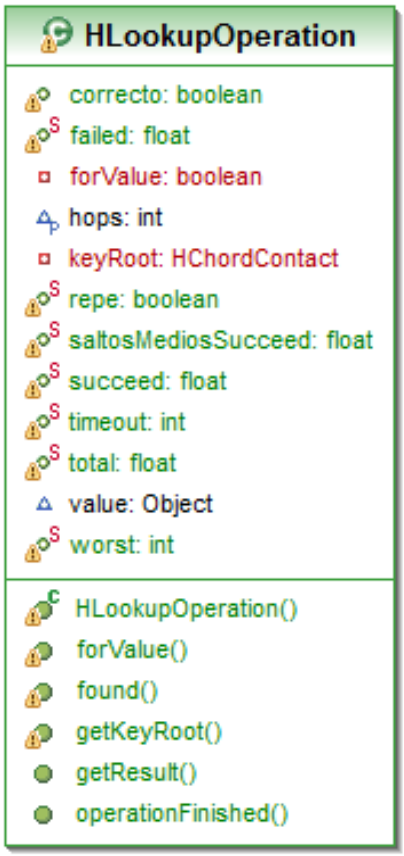


Fig. 33: HLookupOperation

3.5.3.5. Mantenimiento

Una red de ordenadores en la que se intercambian datos entre ellos debe mantener un correcto funcionamiento incluso cuando algunos nodos abandonan la red. Para que esto se lleve a cabo correctamente es necesario ejecutar cuatro operaciones de mantenimiento y actualización.

HCheckPreOperation se encarga de comprobar el nodo al que apunta el puntero del predecesor está aun activo en la red. Un nodo n hace un ping a su predecesor. Si obtiene respuesta, el nodo sigue activo, si no obtiene respuesta, iguala ese campo a null y espera a que se lleve a cabo HStabilizeOperation.

HNotifyOperation es utilizada por los nodos para notificar a sus sucesores que existe un nodo que puede ser su nuevo predecesor. El nodo que recibe este mensaje de notificación es el responsable de comprobar que realmente su predecesor ha cambiado. El nodo sucesor compara el identificador de su predecesor actual con el del nodo que ha mandado la notificación. Si el identificador del nodo que envía el mensaje está entre el sucesor y el predecesor actual, el puntero al predecesor se actualiza.

HStabilizeOperation tiene como principal objetivo mantener actualizados los punteros a los nodos sucesores. Esta operación es una parte importante del proceso de actualización ya que hasta que el nodo predecesor no actualiza su puntero de sucesor al nodo que se une a la red, los mensajes no pueden ser encaminados a través de él.

En la red de interconexión sólo se ejecuta cuando un nodo se une a ella, ya que posteriormente es una red estática debido a que una de las condiciones impuestas al principio del proyecto fue que los super-peers no sufrieran churn.

En el caso de los dominios inferiores, el mecanismo de estabilización se ejecuta siempre que un nodo se une a la red y de forma periódica cada 10 segundos para obtener información sobre posibles nuevos nodos

vecinos.

En Chord y H-Chord un nodo *n* actualiza el puntero al nodo sucesor *s* preguntando a *s* por su predecesor *p*. Si el identificador de *p* está entre los identificadores de *n* y *s*, *n* tiene un nuevo sucesor *p*. Mediante la operación HNotifyOperation, se notifica al nuevo sucesor de que tiene un nuevo predecesor.

HFixfingersOperation es utilizada por todos los nodos de forma periódica cada 10 segundos para asegurar que las entradas de su tabla son correctas. Esta operación no comprueba si un puntero es correcto o no, busca el nodo que corresponde a cada posición de la finger table y le asigna ese valor sin tener en cuenta el que había anteriormente.El $finger^{ith}$ apunta al nodo que es igual o sucede al identificador 2^{i-1}.

La tabla 1 del apartado 2.1.3.1 muestra la creación de una finger table.

Estas operaciones se llevan a cabo tanto en la red de nivel superior como en las redes de nivel inferior. En la red de interconexión se trabaja con el netID y en los dominios, con el id. Sus atributos y métodos se especifican en la figura 34.

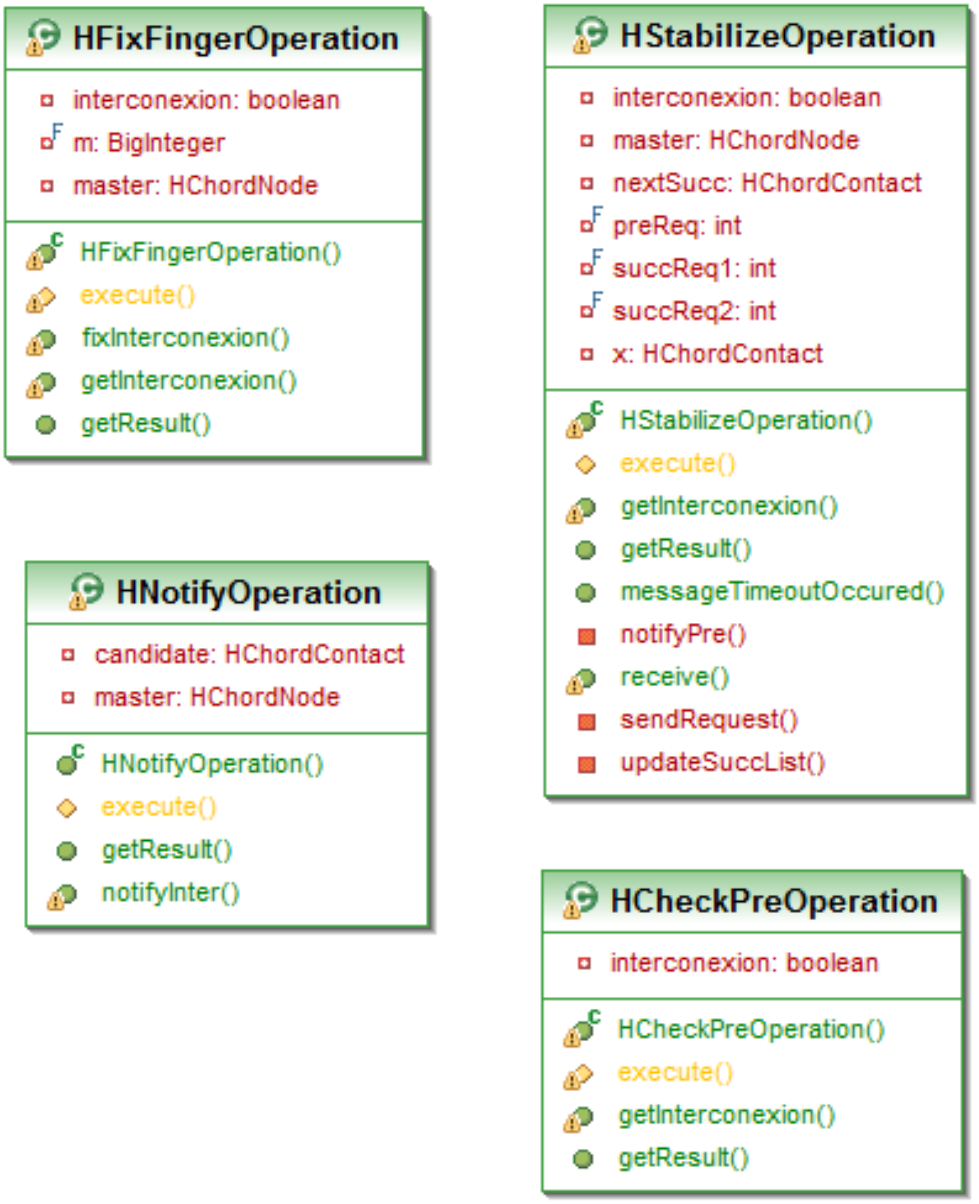


Fig. 34: Operaciones de mantenimiento

En http://enjambre.it.uc3m.es/~rgonzalez/PFC_Sandra_Marco/, puede obtenerse el código del simulador, la implementación de partida, implementación final de la red jerárquica Chord y los programas auxiliares para crear los archivos de los que se habla en los anexos I y II.

Capítulo IV

4. Evaluación

4.1. Aplicaciones externas necesarias para las simulaciones

4.1.1. Configuración del escenario de simulación

El escenario de simulación se configura mediante un archivo XML externo que se pasa como parámetro al simulador. La estructura de este archivo puede verse en el Anexo I, en el que se explican cada una de las partes de las que se compone.

Para la creación de este archivo ha sido necesario crear un pequeño programa en java ya que cada simulación necesita un archivo diferente y debido al gran número de simulaciones que se llevan a cabo, resulta muy tedioso crearlos de uno en uno. La aplicación ha sido programada con Eclipse y finalmente se crea el archivo .jar correspondiente.

4.1.2. Operaciones de la simulación

Todas las operaciones que se llevan a cabo durante la simulación están recogidas y programadas de antemano en un archivo con extensión .dat. Este archivo recoge todos los “join, “leave”, “store” y “valueLookup” que

se ejecutan. Éste se pasa como variable al archivo XML descrito en el apartado anterior. Finalmente se crea el archivo .jar correspondiente.

Para su creación ha sido necesario también implementar un programa en java siguiendo unas determinadas características de operaciones y tiempos dependiendo del tamaño de la red en cada simulación así como de otros parámetros. En el anexo II se muestra la estructura del archivo.

4.1.3. Script

Lanzar una a una todas las simulaciones es un trabajo muy tedioso que se ha evitado creando un script que realiza cuatro pasos principales:

- Creación del archivo de configuración.
- Creación del archivo de operaciones.
- Lanzamiento de la simulación utilizando los dos archivos anteriores.
- Recopilación de la información obtenida de la simulación.

Estos pasos se repiten tantas veces como simulaciones sean necesarias lanzar, modificando los valores de número de peers, número de redes y la probabilidad de búsqueda en el dominio propio (ρ).

4.2. Simulaciones

El routing performance [11], o lo que es lo mismo, el número medio de saltos necesarios para alcanzar el destino vendría descrito por:

$$(a) \qquad RP = D(M) + \frac{K-1}{K}\left[1 + C\left(\sum_{k=1}^{k} S_k\right)\right]$$

donde:

- D(M)=Coste de encontrar un peer en su propio dominio.
- K= número de dominios.
- $C\left(\sum_{k=1}^{k} S_k\right)]$: Coste de encontratar un super-peer en la red de interconexión.

Para validar este modelo se realizan una serie de simulaciones, utilizando el simulador PeerfactSim.KOM, que permiten obtener unos datos y realizar un posterior análisis.

Estas simulaciones se lanzan sobre la máquina Hera de la universidad utilizando 3 máquinas virtuales. Como apoyo, en algunos momentos también se ha utilizado un ordenador personal.

Se ha elegido la virtualización de servidos [15] ya que es una tecnología que permite que varias máquinas virtuales funcionen en el mismo servidor físico. Cada máquina virtual permanece completamente aislada de las otras, y desvinculada del host subyacente mediante una fina capa de software denominada hipervisor.

La plataforma de virtualización elegida ha sido Citrix XenServer basada en el hipervisor Xen. Esta elección se ha tomado debido a buenas

experiencias previas en otros proyectos y a la robustez que ofrece. Xen proporciona aislamiento seguro, control de recursos y garantías de calidad de servicio.

Con el fin de obtener unos valores fiables para poder evaluar el funcionamiento de la red se han llevado a cabo un total de 2470 simulaciones.

Se han ido modificando el número de peers, número de dominios en los que están distribuidos y la probabilidad de que un peer haga una búsqueda en su propio dominio (ρ) o en otro. Con cada combinación posible se han realizado 10 simulaciones. Los posibles valores de los parámetros son:

- Número de peers: {100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000}
- Número de redes: {1, 5, 10, 20}
- Probabilidad ρ: {0.3, 0.6, 0.9, 1/número de redes}

Las simulaciones constan de dos fases.

La primera es un periodo transitorio en el que los peers se unen a la red y almacenan un dato, elegido aleatoriamente, cada uno. En esta fase los peers no muestran el comportamiento habitual ya que los tiempos de entrada en la red no son los expuesto en el apartado 3.3.3 de acuerdo al churn de la red, si no intervalos constantes de tiempo calculados dividiendo el número total de peers entre el total del tiempo transitorio.

En la segunda fase, los peers comienzan a tener el comportamiento esperado de una red Chord uniéndose y dejando la red siguiendo una distribución binomial negativa. En esta fase el número medio de peers en la red es el número de peers que había al finalizar la primera fase.

La primera fase tiene una duración de 30 minutos mientras que la segunda se lleva a cabo durante 120 minutos. Sólo en la segunda fase es cuando se recogen los datos que posteriormente se someterán a estudio.

4.3. Resultados

Una vez acabadas todas las simulaciones lanzadas se analizan los resultados obtenidos con ayuda de Excel. Para ello se crea una tabla dinámica que organiza todos los datos y permite la creación de las gráficas más representativas a través de la incorporación de filtros de valores.

Para cada número de peers, número de dominios y probabilidad de búsqueda en la propia red, se calcula la media y el intervalo de confianza. Cada escenario se ha repetido 10 veces con el fin de mejorar la precisión y obtener una desviación típica muy baja así como un intervalo de confianza al 95% con un error medio menor del 6.4%.

Entre todos los datos obtenidos del experimento, los más interesantes son el número de saltos necesarios para encontrar un peer y el número de entradas en las finger table. A continuación se representan y estudian estos valores.

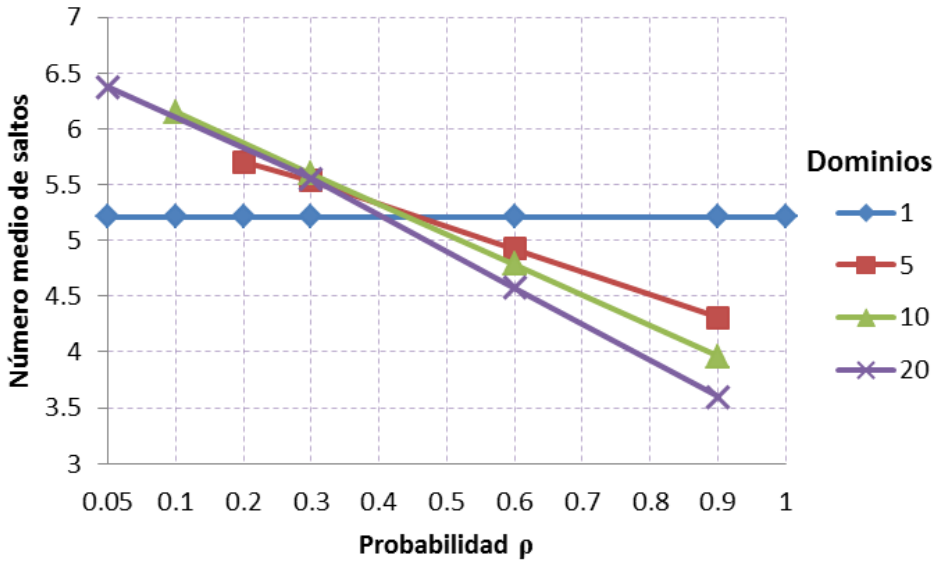


Fig. 35: Número medio de saltos en función de la probabilidad trabajando con 10000 peers

La gráfica de la figura 35 representa el número medio de saltos necesarios para alcanzar un peer en la red Chord según el número de dominios en los que están organizados y la probabilidad de que el peer destino se encuentre en el mismo dominio que el peer origen. Se puede observar que cuando mayor es la probabilidad de buscar un peer en el propio dominio, más clara es la ventaja que se obtiene de tener varios dominios. También se deduce que para probabilidades mayores de aproximadamente 0.4 es conveniente trabajar con varios dominios, y cuantos más, mejor.

Este resultado es totalmente lógico ya que a medida que aumenta el número de dominios disminuye el número de peers en cada dominio. Si la probabilidad de búsqueda en el propio dominio es alta, ésta se lleva a cabo entre un grupo de peers más reducidos y el número de saltos hasta llegar al destino será menor. En el caso de probabilidades bajas, es necesario acceder más veces a la red de interconexión, la cual será más grande según aumenta el número de dominios.

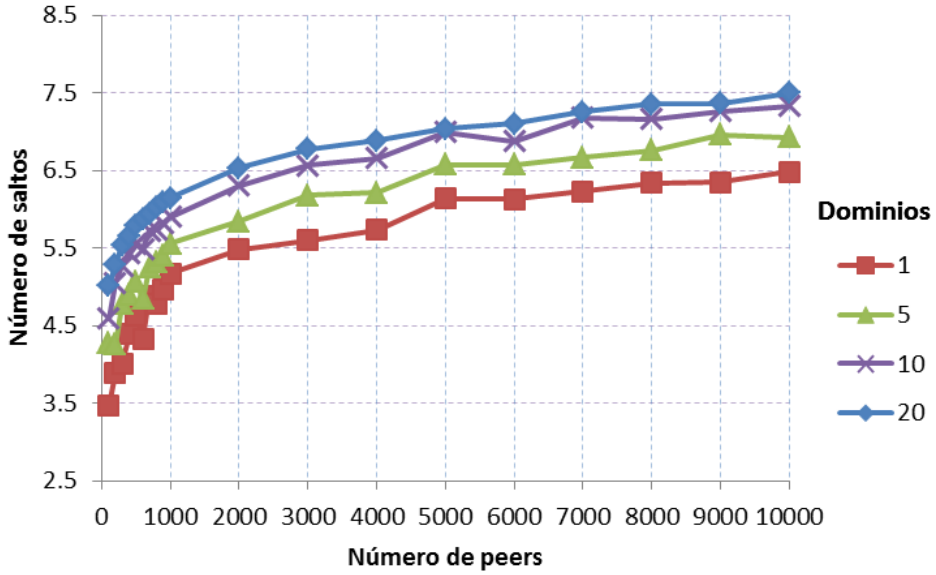
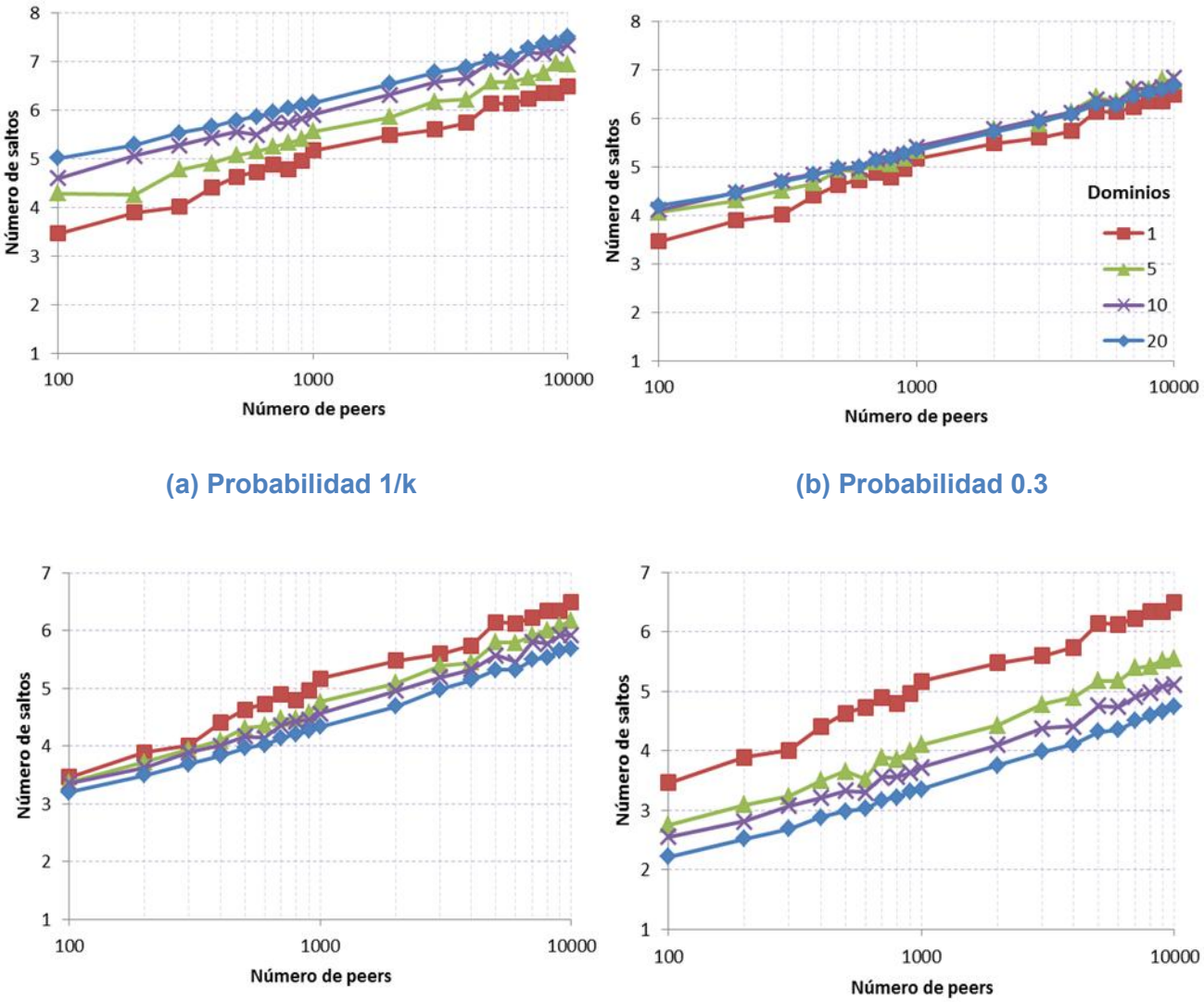


Fig. 36: Número medio de saltos con probabilidad 1/k

La gráfica de la figura 36 muestra el número medio de saltos necesarios para que una query alcance su objetivo cuando la probabilidad de que el peer destino esté en la misma red que el origen es 1/k siendo k el número de dominios.

Lo que resulta interesante de está gráfica con el eje de abscisas en unidades naturales es, que el incremento del número de saltos es mucho más rápido para valores pequeños de número de peers que para valores. En los primeros valores, la pendiente es muy acusada.

Para el caso de 1 dominio, pasa de 3,5 saltos a 5 en apenas 1000 peers, mientras que entre 6000 y 7000 peers esta variación apenas es de 1 o 2 décimas.



(a) Probabilidad 1/k

(b) Probabilidad 0.3

(c) Probabilidad 0.6

(d) Probabilidad 0.9

Fig. 37: Número medio de saltos para distintas probabilidades

La figura 37 muestra 4 gráficas que representan, igual que la figura 36, el número medio de saltos pero está vez la escala del eje de abscisas es logarítmica y en cada gráfica hay una probabilidad, de esta forma se ve el efecto que esto produce.

Se comprueba de forma muy clara que según aumenta el número de peers en la red, manteniendo constante el número de dominios, aumenta también el número de saltos y lo hace aproximadamente en la misma proporción sea cual sea el número de super-peers. Este incremento es lineal en escala logarítmica. En el caso de que se trabaje con un único dominio la curva es igual en todas las gráficas ya que la probabilidad es siempre la misma, 1.

En la gráfica 37.(a) la probabilidad es 1/k, es decir, el peer destino puede estar en cualquiera de los k dominios con la misma probabilidad. En este caso trabajar con 5, 10 o 20 dominios no sale muy rentable ya que el número de queries que atraviesan la red de interconexión es mucho mayor que el número de queries que se quedan en el dominio del peer de origen. Utilizar 1 dominio sería la opción más apropiada.

En la gráfica 37.(b) se trabaja con una probabilidad de 0.3. Sigue sin ser óptimo utilizar muchos dominios debido a que el coste de atravesar una red de interconexión aun no sale rentable. Se comienza a observar la tendencia comentada en la figura 35 en la que se dice que a partir de aproximadamente una probabilidad de 0.4, la mejor opción es contar con varios dominios.

La gráfica 37.(c) representa más claramente las ventajas de trabajar con varios dominios cuando las probabilidades son altas. La curva de 1 dominio se mantiene como en las 2 gráficas anteriores pero el número de saltos medios para 5, 10 y 20 dominios ha disminuido considerablemente debido a que el número de queries que atraviesan la red de interconexión sigue disminuyendo.

En la gráfica 37.(d) continúa la disminución de número de saltos y además se comprueba que cuantos más dominios se utilicen, mejores son los resultados. Con una probabilidad de 0.9, sólo el 10% de las queries atraviesan la red de interconexión ya que tienen como destino un peer perteneciente a otro dominio. El número de saltos necesarios en una red de

10000 peers utilizando 20 dominios se ve reducido a 4.7 en contraste con los 6.5 necesarios si se trabaja con un único dominio.

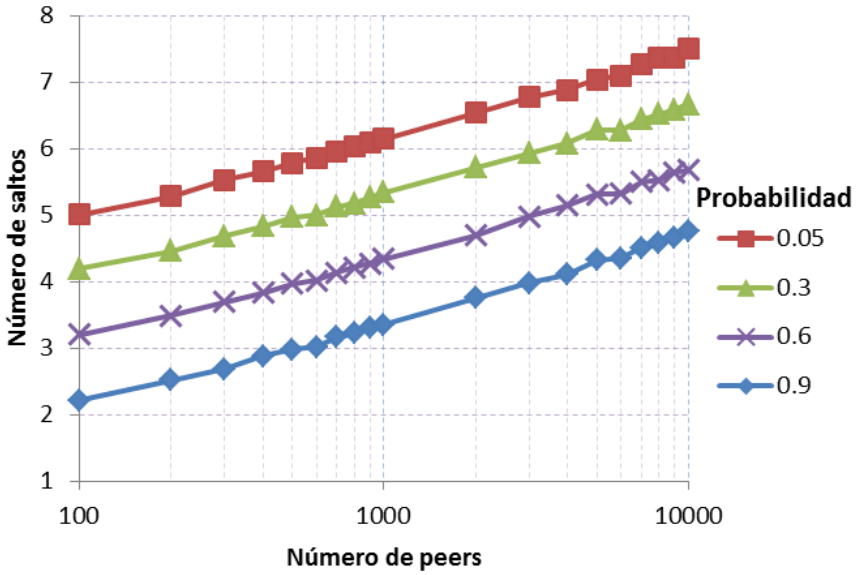


Fig. 38: Número medio de saltos con 20 dominios

Si para el caso de 20 dominios se diferencian las gráficas según la probabilidad se obtiene la representación de la figura 38. En ella se ve claramente la tendencia a disminuir el número de saltos cuando la probabilidad de buscar en la propia red aumenta. Cuanto mayor sea el número de búsquedas que atraviesan la red de interconexión mayor es el número de saltos.

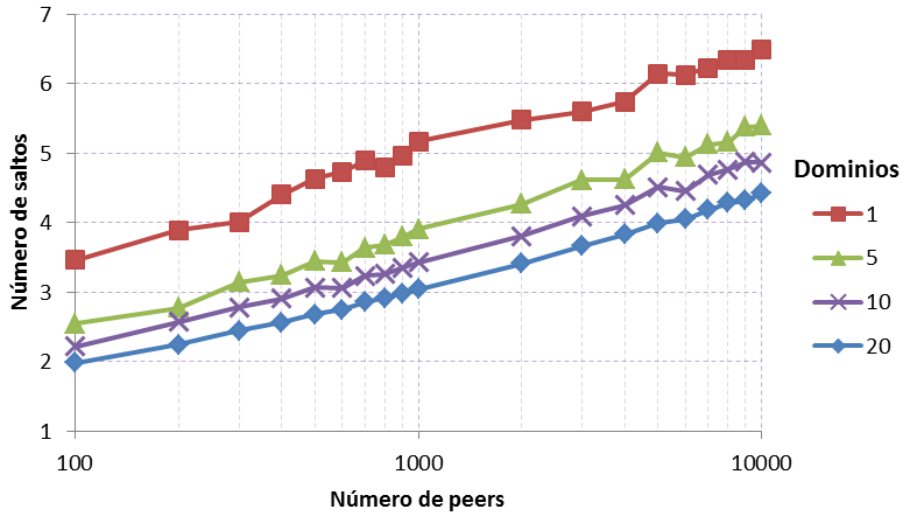


Fig. 39: Número medio de saltos búsquedas intra-dominio

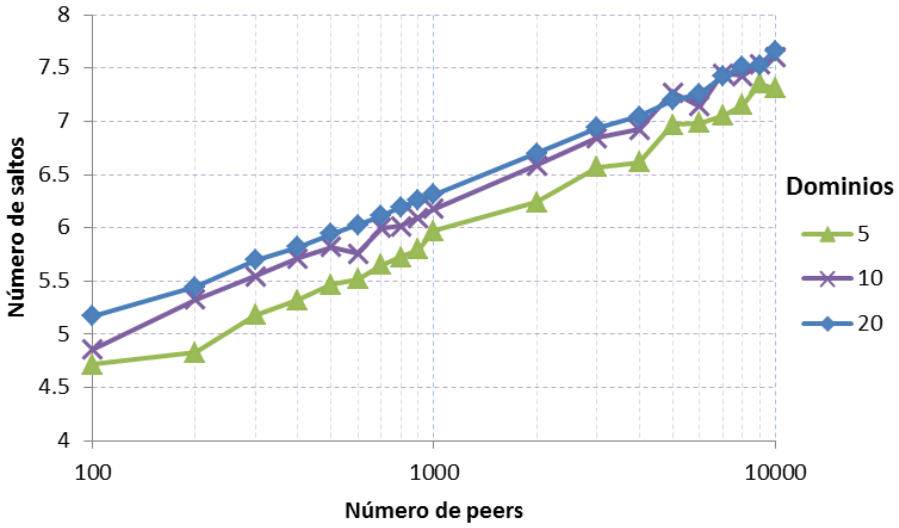


Fig. 40: Número medio de saltos en búsquedas inter-dominio

Separando las queries intra-dominio y las queries inter-dominio, se puede estudiar el efecto de estos dos tipos de búsquedas por separado. Estos datos quedan representados en las figuras 39 y 40. Las conclusiones que se obtienen son totalmente lógicas:

- Según aumenta el número de peers, aumenta el número de saltos. Cuanto más grande es la red, hay que realizar la búsqueda entre más peers y más saltos serán necesarios.

- Cuantos más dominios haya, menor será el número de saltos en las queries intra-dominio. Esto se debe a que al haber más dominios, hay menos peers en cada dominio y la búsqueda se realiza en un entorno más reducido.

- Cuantos más dominios hay, mayor será el número de saltos en las queries inter-dominio. Al haber más dominios la red de interconexión es más grande y serán necesarios más saltos para encontrar al super-peer responsable del dominio del peer destino.

Otro punto importante a tratar a la hora de evaluar el funcionamiento de una red es el número de entradas que tienen las tablas de rutas de los peers. En esta red jerárquica, los super-peers tienen dos finger tables, una para la red de interconexión y otra para el dominio al que pertenecen. En las gráficas siguientes se exponen los puntos más interesantes.

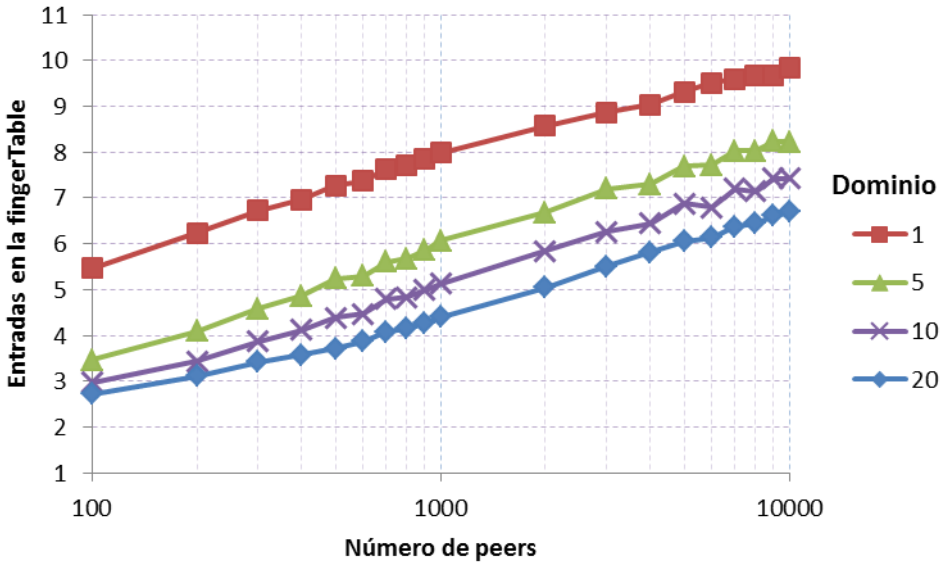


Fig. 41: Entradas en la Finger table intra-dominio

La 41 muestra, en función del número de peers y para distintas cantidades de dominios, el número medio de entradas no repetidas en la finger table normal, es decir, la que se utiliza a la hora de envía mensajes en el dominio propio.

Cuantos más peers haya en la red mayor será el número de entradas que tiene la finger table de cada peer con el objetivo de que el funcionamiento siga siendo el mejor posible. Como puede observarse, la evolución es lineal en escala logarítmica

Cuantos más dominios existen, menos peers hay en cada dominio y por tanto un peer necesita guardar información de menos peers, así las finger tables normales tendrán menos entradas.

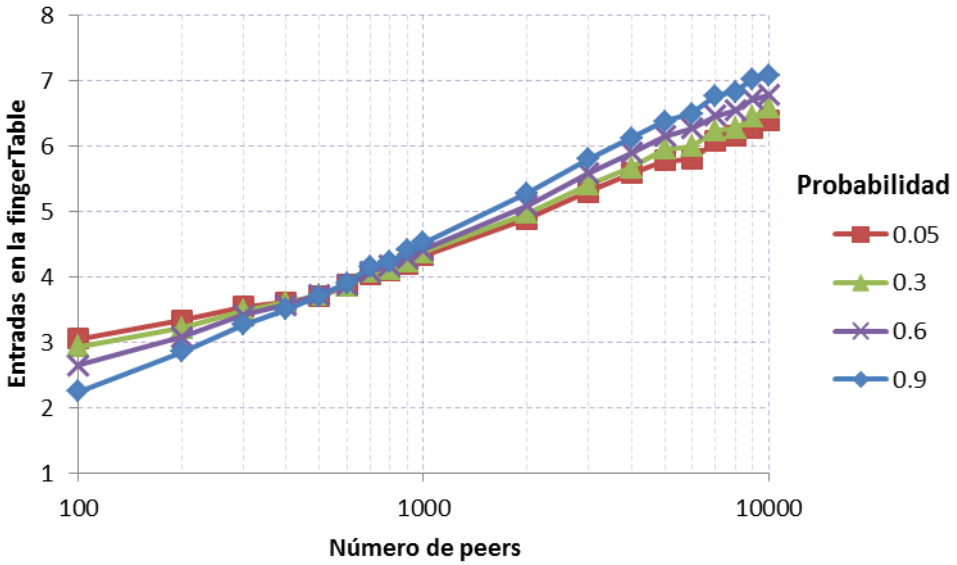


Fig. 42: Entradas en la Finger table intra-dominio con 20 dominios

En la figura 42 se observa una pequeña dependencia del número medio de entradas en la finger table con la probabilidad de búsqueda en el dominio propio. Si esta probabilidad toma un valor alto, las queries intra-dominio son más habituales y las finger table normales están ligeramente más pobladas para valores altos de peers.

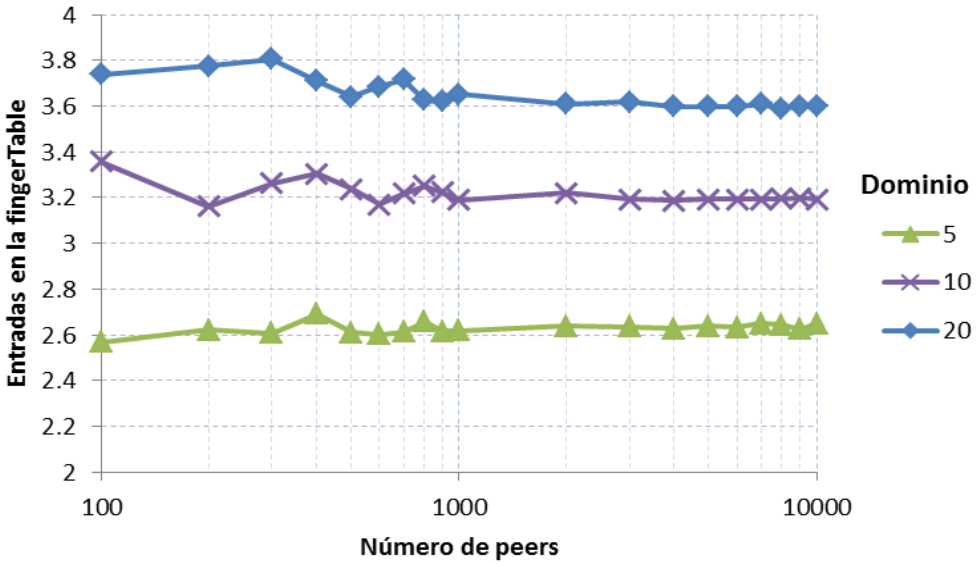


Fig. 43: Entradas en la Finger table de interconexión

La figura 43 representa el número medio de entradas no repetidas en la finger table de interconexión. Puede observarse que para 1 dominio no hay datos ya que en ese caso no existe ningún super-peer y no existe red de interconexión.

El número de entradas se mantiene constante (como mucho hay una diferencia de 1 décima que puede despreciarse) independientemente del número total de peers que conformen la red. Esto se debe a que la finger table de interconexión sólo depende del número de dominios y es indiferente al número de peers que conformen cada dominio. Cuantos más dominios hay, más grande es la red de interconexión y los super-peers guardan información de más vecinos.

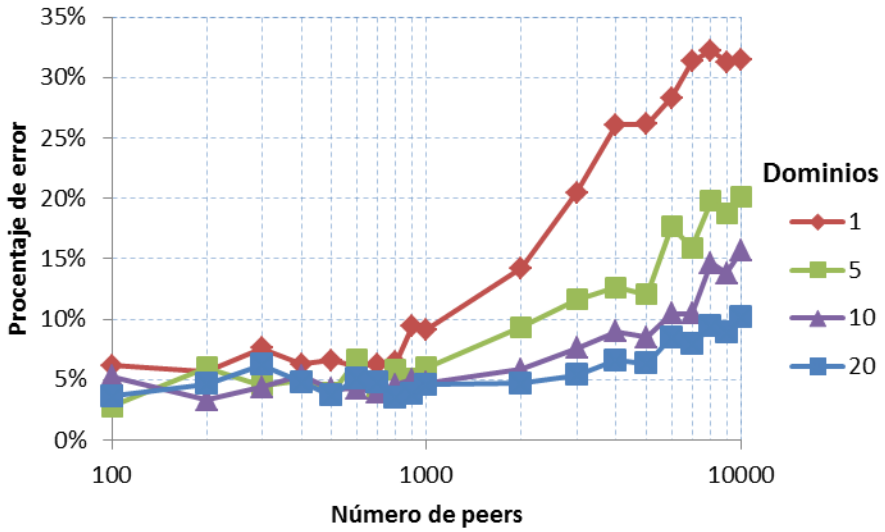


Fig. 44: Porcentaje de queries fallidas

Un problema importante que se tenía con el código original con el que se empezó a trabajar era el elevado número de queries que no finalizaba o no lo hacían correctamente. Se introdujo, como se ha dicho en el apartado 3.4, un mecanismo de replicación y el reintento en caso de fallo y los resultados mejoraron, aun así, si se trabaja con un solo dominio el porcentaje de error es demasiado alto para redes de un solo dominio. Estas dos modificaciones en la red plana se mantienen a la hora de trabajar con 5, 10 y 20 dominios.

La figura 44 muestra este porcentaje de error. Quizá, la reducción de errores sea uno de los mejores beneficios que se obtienen de trabajar con más de un dominio. Para redes con un número bajo de peers, los errores que se comenten no son excesivamente elevados, pero según aumenta el número de peers, este valor se dispara si se distribuyen en un único dominio. Según aumenta el número de dominios, el porcentaje de errores disminuye.

Capítulo V

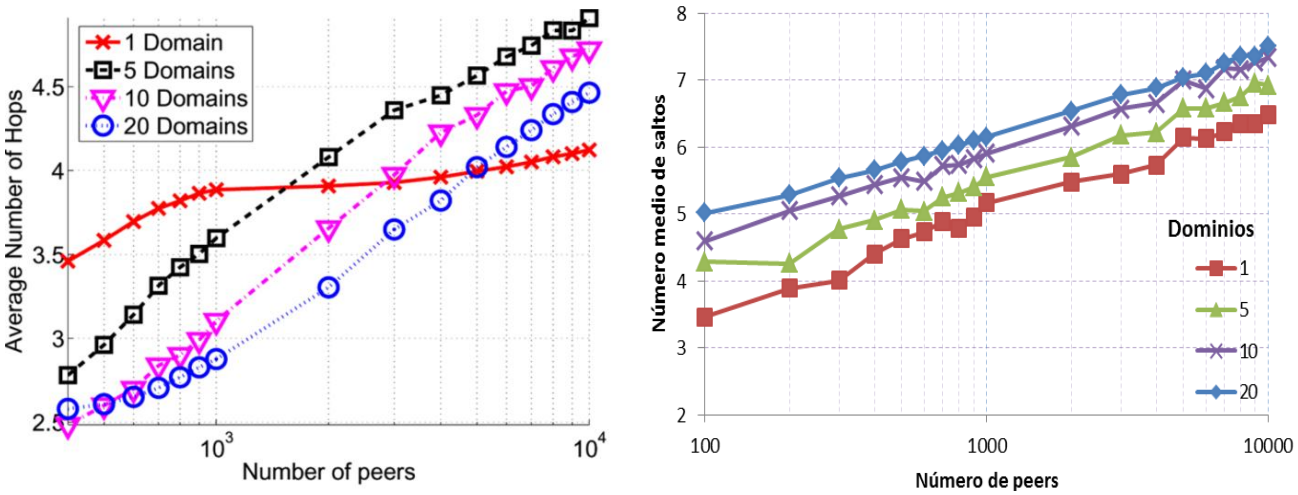
5. Comparación con la red Kademlia

A la hora de trabajar con una red peer-to-peer resulta interesante tener varias opciones y compararlas entre ella para saber cual encaja mejor con los propósitos marcados. Las distintas distribuciones de peers en las redes y los distintos protocolos de encaminamiento pueden influir a la hora de tomar la decisión.

A continuación se muestra una comparación de los resultados obtenidos en este documento con los de una red Kademlia [14]. Lo nuevos datos a estudio se han obtenido utilizando Kademlia tanto en la red de interconexión como en todos los dominios que la conforman. Las pruebas realizadas para obtener esos resultados fueron las mismas que las realizados en este proyecto con la red Chord y el simulador sobre el que se lanzan las simulaciones también es el mismo aunque en una versión anterior.

En el apartado 2.1.3.2 de este documento se hace una introducción al funcionamiento y principales características de Kademlia.

El objetivo de ambos experimentos ha sido encontrar una relación entre el número medio de saltos necesarios para que concluya con éxito una búsqueda y el número medio de entradas en la tabla de rutas en función del número de peers que conforman la red y el número de dominios en que están distribuidos.

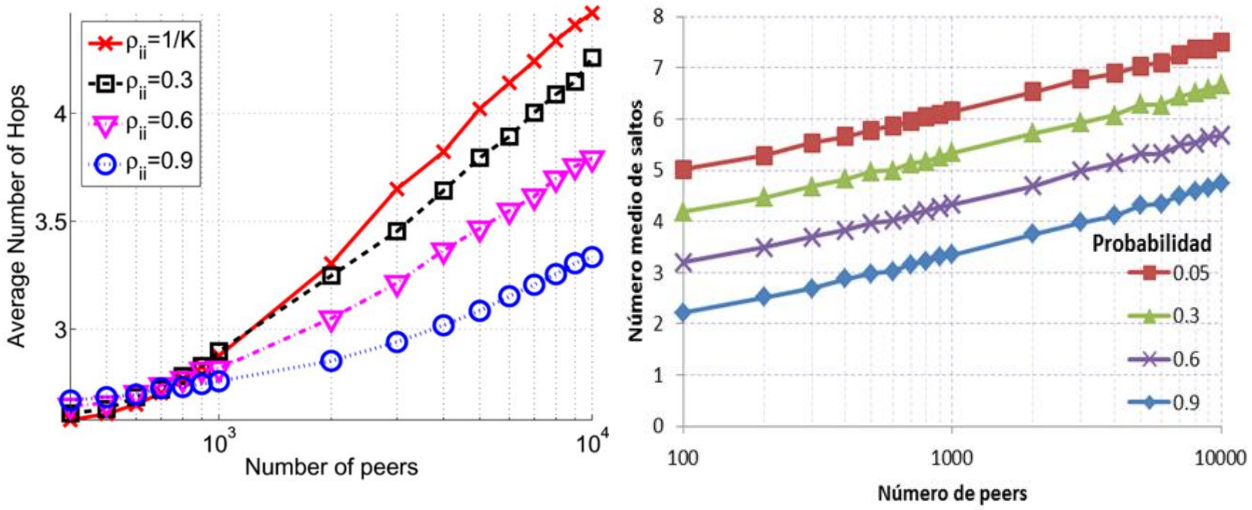


(a) Kademlia (b) Chord

Fig. 45: Número medio de saltos en Kademlia y Chord para operaciones de value-lookup

La figura 45 ilustra el número medio de saltos necesarios en una operación de valueLookup para 1, 5, 10 y 20 dominios cuando la probabilidad de búsqueda en el dominio propio es 1/k, es decir, la inversa del número de dominios. La figura 45.(a) corresponde a la red Kademlia y la figura 45.(b) a la red Chord.

Se observa que para el caso en el que las queries se distribuyen con la misma probabilidad entre todos los dominios que conforman la red, Kademlia lanza unos resultados mejores que Chord con el aumento del número de dominios. En el caso de Chord, el crecimiento de la red de interconexión produce un aumento muy acusado en el número de saltos cuando la probabilidad de búsqueda en el propio dominio es baja.



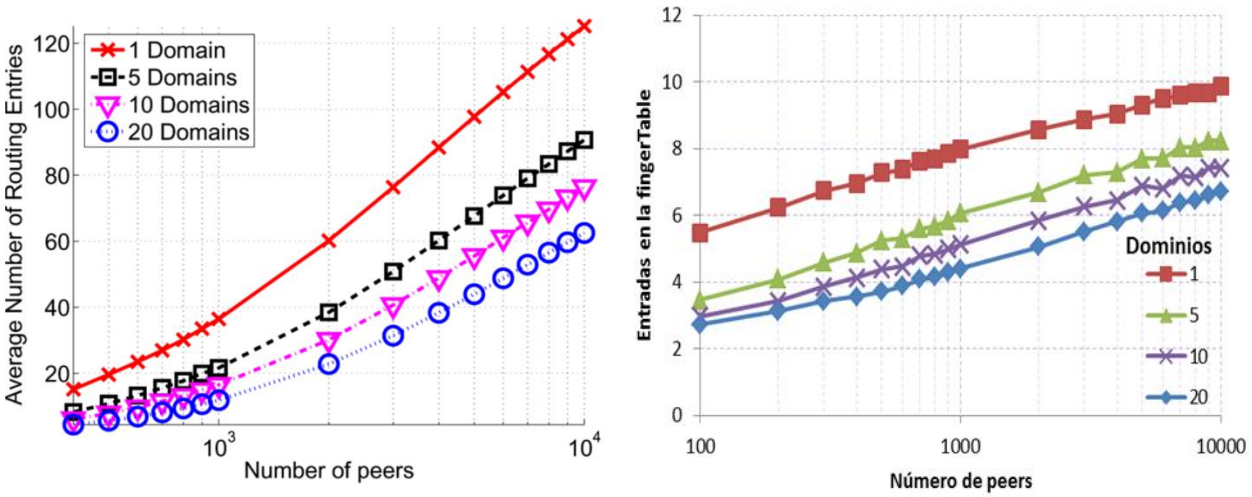
(a) Kademlia (b) Chord

Fig. 46: Número medio de saltos utilizando 20 dominios para operaciones de value-lookup

La 46 muestra el número medio de saltos para 20 dominios en las redes Chord y Kademlia. Se pueden sacar varias conclusiones de estas gráficas.

- El número medio de saltos es mayor en la red Chord que en Kademlia sea cual sea el número de peers y la probabilidad ρ. Esto indica que la escalabilidad de Chord es peor.

- Las dos redes ofrecen distinto comportamiento al comparar las curvas de mismas probabilidades. El incremento del número de saltos según aumenta el número de peers es constante en el caso de Chord para distintas probabilidades mientras que en Kademlia este incremento es más acusado cuando menor es la probabilidad ρ.



(b) Kademlia

(a) Chord

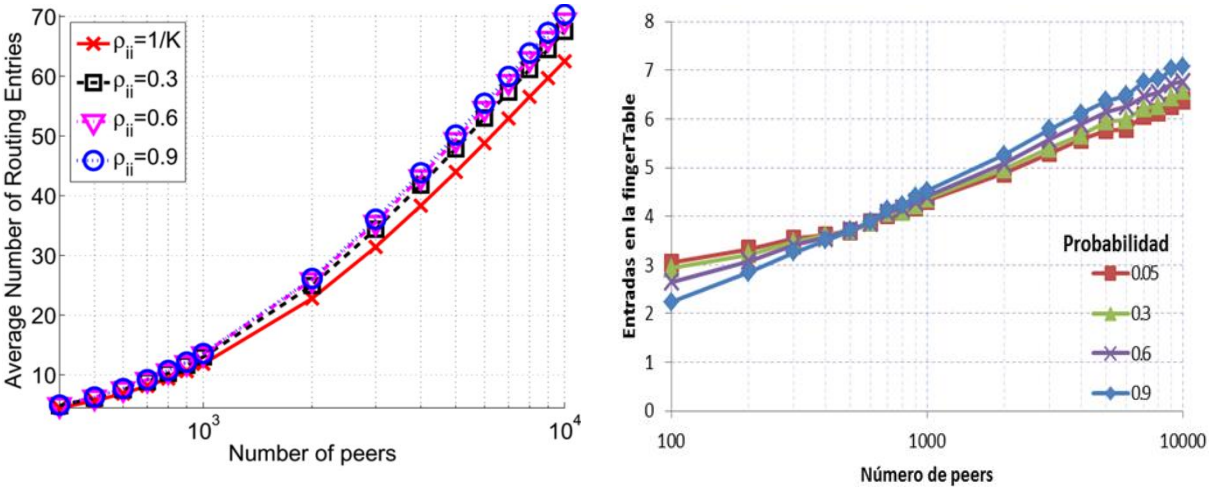
Fig. 47: Número medio de entradas en la tabla de rutas

El número medio de entradas en la tabla de rutas es uno de los parámetros interesantes a estudiar. Es importante tener en cuenta que las tablas de rutas en ambas redes no se crean ni funcionan de la misma manera. En 2.1.3 se puede encontrar una breve explicación sobre los dos tipos.

En la figura 47 se observa que tanto en Chord como en Kademlia existe una dependencia entre el número de dominios y el número de entradas en la tabla de rutas. Cuantos más dominios conforman la red, menos peers hay en cada dominio y menos entradas. También se ve claramente, observando los valores del eje de ordenadas, que Chord guarda información de muchos menos peers que Kademlia. Otro punto destacable es que el incremento del número de entradas es mucho menos acusado en Chord según aumenta el número de peers que conforman la red.

Relacionando los datos de las figuras 46 y 47, se entiende por lógico que la red Chord sea menos eficiente en cuando al número medio de saltos debido a que sus peers almacenan información de mucho menos nodos de su dominio que en Kademlia. Kademlia necesita menos saltos para que un mensaje llegue a su destino, pero a cambio, el tráfico generado para el

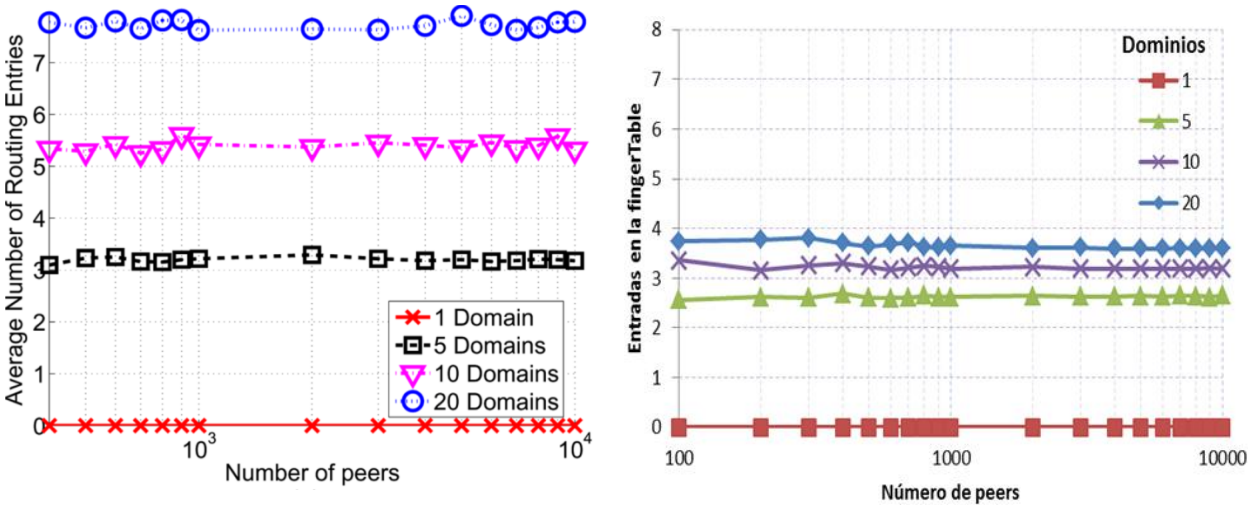
mantenimiento y actualización de sus punteros será mucho más elevado que en Chord.



(a) Kademlia (b) Chord

Fig. 48: Número medio de entradas en la finger table cuando se trabaja con 20 dominios

En las dos gráficas de la figura 48 se observa una leve dependencia entre ρ y el número de entradas en la tabla de rutas. Tanto en Chord como Kademlia, para valores altos de ρ, las queries intra-dominio son más probables y las tablas de rutas están ligeramente más pobladas.



(a) Kademlia (b) Chord

Fig. 49: Número medio de entradas en la finger table de interconexión

Como es de esperar, el número medio de entradas en las tablas de rutas de interconexión, tanto en Chord, como Kademlia, es el mismo independientemente del número de peers que haya en la red. De la misma manera que ocurre en las tablas de rutas que contienen información intra-dominio, en las de interconexión, el número de entradas es mayor en Kademlia que en Chord ya que el mecanismo de funcionamiento y mantenimiento de ambas es el mismo.

Capítulo VI

6. Conclusiones y trabajos futuros

6.1. Conclusiones

En este proyecto se ha implementado una red Chord que permite el intercambio de información entre peers organizados de forma jerárquica en una overlay lógica. Para ello se han realizado una serie de modificaciones en la red plana que han mejorado su rendimiento y se han añadido funcionalidades para cumplir los requisitos de la nueva estructura de red.

Los peers se organizan en dominios y se crea una red de interconexión que los une para que puedan intercambiarse información. Esta red de interconexión está formada por super-peers, los cuales realizan también las funciones de un peer normal. Para saber si se tiene que trabajar sobre un dominio o la red superior, se crean los identificadores jerárquicos compuestos por dos partes, el netID, identifica el dominio y el ID identifica el peer o el recurso almacenado.

Una vez que se han obtenido y estudiado los resultados se obtienen una serie de conclusiones respecto a estas redes:

- **Mejora del rendimiento.** La creación de pequeños grupos de peers interconectados entre sí ha llevado a una disminución del número de saltos necesarios para alcanzar el destino cuando la probabilidad de búsqueda en el dominio propio es mayor que la probabilidad de búsqueda en otro dominio. La estructura jerárquica de Chord es eficiente cuando los nodos se pueden ordenar en grupos, atendiendo a alguna característica, de manera que el número de consultas que realicen sea mayor entre los peers del mismo grupo.

- **Más carga en los super-peers.** Por estos peers, a parte del tráfico normal de cada dominio, pasan todos los mensajes cuyo destino está en otro dominio. Esto hace que la carga que soportan sea mayor que en el resto pero no tanta como para influir en el funcionamiento de la red.

- **H-Chord vs H-Kademlia.** Tras el estudio comparativo de las dos redes se concluye que H-Kademlia tiene una mayor eficiencia que H-Chord para valores pequeños de la probabilidad de búsqueda en el dominio propio. Por otra parte, los peers en H-Chord almacenan menos información acerca de la composición de la red que los peers en H-Kademlia.

- En el ámbito personal este proyecto ha supuesto una experiencia enriquecedora y una primera toma de contacto con lo que pueden llegar a ser la exigencias del mundo laboral. A los conocimientos adquiridos sobre el tema que trata propiamente dicho hay que añadir los relacionados con las herramientas utilizadas para la consecución de los objetivos y la dificultad de trabajar con código programado por otras personas.

- Se han alcanzado los objetivos propuestos inicialmente excepto la intención de que tanto la red de interconexión como los dominios pudieran ser indistintamente Chord o Kademlia. Esta idea se desechó debido a problemas con el código de partida de Kademlia del que disponíamos.

6.2. Trabajos futuros

El estudio realizado en este proyecto no cierra, ni mucho menos, las puertas a seguir investigando en el campo de las redes peer-to-peer. Durante la realización del mismo, han ido surgiendo ideas sobre mejoras y posibles futuros trabajos siguiendo la línea de este proyecto.

- **Introducción del churn en los super-peers.** En este proyecto los super-peers son fijos y se ha trabajado sin la posibilidad de que puedan abandonar la red de forma voluntario o debido a un fallo. Resultaría interesante que estos se comportaran como peers normalmente en cuanto a churn se refiere y que pudieran ser elegidos de forma dinámica. Debería existir un mecanismo según el cual, si un super-peer abandona la red, un peer de ese mismo dominio fuera capaz de desarrollar ese trabajo de forma inmediata para no romper el funcionamiento natural de una red jerárquica.

- **Realizar este mismo trabajo con otros protocolos P2P.** Ya que los resultados obtenidos al comparar una red jerárquica con su correspondiente red plana son satisfactorios, sería interesante poder comparar los resultados obtenidos para Kademlia y Chord con otros protocolos P2P como puede ser CAN. Para ello sería necesario implementar las redes jerárquicas de esos protocolos, lanzar las simulaciones correspondientes y recoger los mismos datos que se han recogido en esta memoria. De esta forma se podría realizar un estudio comparativo más completo.

- **Unir redes con distintos protocolos.** El siguiente paso consistiría en trabajar con distintos protocolos en los distintos dominios así como poder elegir el protocolo de la red de interconexión. Una vez hecho esto se podrían elegir los protocolos más adecuados en cada caso teniendo en cuenta distintos usos y requerimientos de la red. También es interesante poder intercambiar información entre grupos de

ordenadores cuya estructura y protocolo de encaminamiento son distintos.

- Por último, habría que realizar **pruebas en entornos reales** para comprobar el funcionamiento real de todos los puntos sobre los que se ha hablado anteriormente.

Anexo I

Configuración del escenario de simulación

A continuación se muestra un ejemplo de un archivo que configura el simulador:

```
<?xml version="1.0" encoding="utf-8" ?>
- <Configuration>
  - <Default>
      <Variable name="seed" value="0" />
      <Variable name="size" value="2000" />
      <Variable name="finishTime" value="9000000000" />
      <Variable name="actions" value="config/dataSandra.dat" />
    </Default>
    <SimulatorCore class="de.tud.kom.p2psim.impl.simengine.Simulator" static="getInstance" seed="$seed" finishAt="$finishTime" />
  - <NetLayer class="de.tud.kom.p2psim.impl.network.simple.SimpleNetFactory" downBandwidth="200" upBandwidth="100">
      <LatencyModel class="de.tud.kom.p2psim.impl.network.simple.SimpleStaticLatencyModel" latency="10" />
    </NetLayer>
    <TransLayer class="de.tud.kom.p2psim.impl.transport.DefaultTransLayerFactory" />
    <ComponentFactory class="de.tud.kom.p2psim.impl.overlay.dht.hierarchical.HNodeFactory" port="400" numOverlays="10" />
  - <Monitor class="de.tud.kom.p2psim.impl.common.DefaultMonitor" start="0" stop="$finishTime">
      <Analyzer class="de.tud.kom.p2psim.impl.analyzer.DHTAnalyzer" />
    </Monitor>
  - <HostBuilder class="de.tud.kom.p2psim.impl.scenario.DefaultHostBuilder" experimentSize="$size">
    - <Host groupID="peer0">
        <NetLayer />
        <TransLayer />
        <ComponentFactory red="0" esSuperpeer="true" tipoRed="chord" />
        <Properties enableChurn="false" />
      </Host>
    - <Host groupID="peer1">
        <NetLayer />
        <TransLayer />
        <ComponentFactory red="0" esSuperpeer="false" tipoRed="chord" />
        <Properties enableChurn="false" />
      </Host>
    - <Host groupID="peer2">
        <NetLayer />
        <TransLayer />
        <ComponentFactory red="0" esSuperpeer="false" tipoRed="chord" />
        <Properties enableChurn="false" />
      </Host>
    </HostBuilder>
  - <ChurnGenerator class="de.tud.kom.p2psim.impl.churn.DefaultChurnGenerator" start="69200000000" stop="$finishTime">
      <ChurnModel class="de.tud.kom.p2psim.impl.churn.ChordChurnModel" />
    </ChurnGenerator>
  - <Scenario class="de.tud.kom.p2psim.impl.scenario.CSVScenarioFactory" actionsFile="$actions"
      componentClass="de.tud.kom.p2psim.impl.overlay.AbstractOverlayNode">
      <ParamParser class="de.tud.kom.p2psim.impl.overlay.dht.chord.hierarchical.HOverlayKeyParser" />
      <ParamParser class="de.tud.kom.p2psim.impl.overlay.dht.chord.hierarchical.HDHTObjectParser" />
    </Scenario>
</Configuration>
```

Fig. 50: Archivo de configuración del escenario de simulación

Este archivo está formado por distintas partes que configuran cada uno de los niveles que se muestran en la figura 51 del anexo III.

En las primeras líneas se da valor a una serie de variables que se utilizaran más adelante. Quedan definidos, por ejemplo, el número de peers que formará parte de la simulación, la duración de la misma, así como el archivo en el que se encuentran las operaciones que se llevan a cabo a lo largo de la simulación.

Continuando con el archivo se llega a la orden que indica cual es la clase que se encarga de preparar el escenario para poder lanzar una simulación. En este caso es "de.tud.kom.p2psim.impl.simergine.Simulator" y necesita 3 parámetros para que funcione correctamente.

```
<SimulatorCoreclass="de.tud.kom.p2psim.impl.simengine.Simulator"
static="getInstance" seed="$seed" finishAt="$finishTime" />
```

Después se indican de la misma forma, las clases que tendrá que utilizar el simulador para configurar la capa de red, la capa de overlay, crear los distintos peers, configurar el churn y obtener todas las acciones del archivo .dat del que se habla en el apartado 4.1.2 de la presente memoria.

Mención especial a las líneas en las que se crean los peers, en las que quedan definidos el nombre del peer, la subred a la que pertenece, si es un superpeer o un peer normal, el tipo de red (chord, kademlia, can…) y si soporta churn o no:

```
<Host groupID="peer0">
<NetLayer />
<TransLayer />
<ComponentFactory red="0" esSuperpeer="true" tipoRed="chord" />
<Properties enableChurn="false" />
</Host>
```

Anexo II

Archivo de operaciones

Los archivos de operaciones son muy largos, en el caso de redes de 10.000 peers, tienen más de 200.000 líneas. A continuación se muestran tres partes de un archivo correspondiente a una red de 100 peers distribuidos en 5 redes.

```
peer0 12000000 join callback
peer0 52000000 store 00@0 00@0 callback
peer21 24000000 join callback
peer21 64000000 store 01@1 01@1 callback
peer42 36000000 join callback
peer42 76000000 store 02@2 02@2 callback
peer63 48000000 join callback
peer63 88000000 store 03@3 03@3 callback
peer84 60000000 join callback
peer84 100000000 store 04@4 04@4 callback
peer15 72000000 join callback
peer15 112000000 store 05@0 05@0 callback
peer45 84000000 join callback
peer45 124000000 store 06@2 06@2 callback
peer30 96000000 join callback
peer30 136000000 store 07@1 07@1 callback
peer101 108000000 join callback
peer101 148000000 store 08@4 08@4 callback
peer83 120000000 join callback
peer83 160000000 store 09@3 09@3 callback
peer18 132000000 join callback
peer18 172000000 store 010@0 010@0 callback
.
.
.
peer90 3650720115 valueLookup 039@1 callback
peer55 3654320114 valueLookup 062@3 callback
peer96 3657769358 join callback
peer52 3657920113 valueLookup 042@4 callback
peer32 3661520112 valueLookup 093@0 callback
.
.
```

.

```
peer26 4248431257 valueLookup 093@0 callback
peer25 4252031256 valueLookup 055@2 callback
peer75 4253154833 leave callback
peer48 4255631825 valueLookup 025@1 callback
peer102 4259231824 valueLookup 040@4 callback
```

El archivo se divide en dos partes bien diferenciadas. La primera corresponde a la creación de la red y el almacenaje de datos en ella por parte de cada peer. En la segunda parte, se realizan las queries y se activa el churn.

La primera columna muestra el peer que realiza la acción. Este peer es elegido aleatoriamente entre los que hay activos en ese instante. Esto se debe a que las primeras operaciones de join corresponden a los superpeers, es decir, primero se crea la red de interconexión y a continuación de van uniendo peers a las redes de nivel inferior de forma aleatoria.

La segunda columna corresponde al tiempo en el que el peer realiza la acción. Estos tiempos son calculados por distintas distribuciones estadísticas dependiendo de la operación que se lleve a cabo. Para las operaciones de join y leave en la segunda parte del archivo, se ha utilizado la distribución binomial negativa**,** variando la media según varía el número de nodos que forman la red [10]. Los tiempos obtenidos para las operaciones de valueLookup siguen una distribución de Poisson. Se realizan, en media, 20 operaciones de valueLookup por nodo.

La tercera columna informa sobre la operación a realizar pudiendo ser estas join, store, leave y valueLookup. Las operaciones de store y valueLookup necesitan, a continuación, un parámetro que indica el dato que se almacenará o buscará según corresponda.

Anexo III

Simulador PeerfactSim.KOM

1. Arquitectura del simulador

La arquitectura básica de PeerfactSim.KOM [16] está dividida en dos partes: las capas de funcionalidad y el motor de simulación. La figura 51 representa estas dos partes.

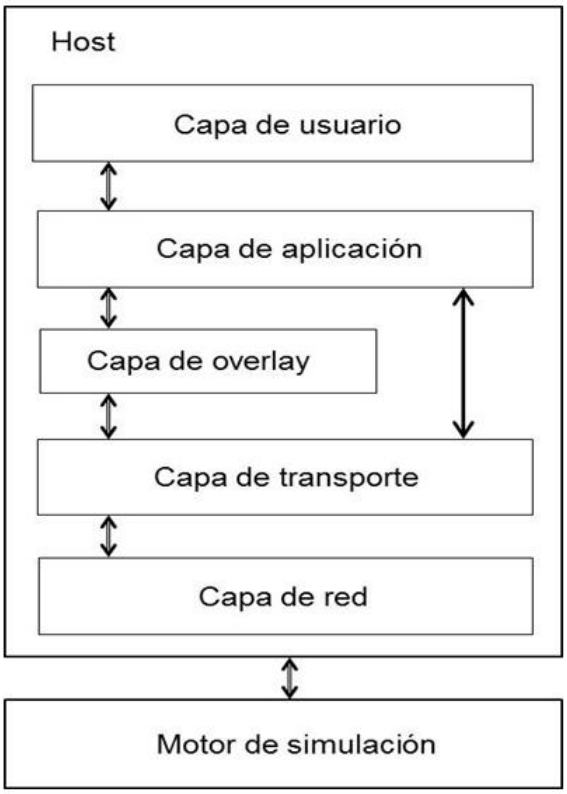


Fig. 51: Arquitectura del simulador

Cada capa realiza un conjunto de funciones que le corresponden. Los componentes básicos de cada nivel ofrecen diferentes servicios que son utilizados por otros niveles o componentes.

La **capa de red** encapsula los detalles de red de la red subyacente. Describe las funcionalidades de las tres primeras capas del modelo OSI bajo una misma interfaz. Los componentes de esta capa están dentro del paquete,

de.tud.kom.p2psim.api.network

Es posible seleccionar fácilmente distintos modelos de servicios de red para variar el grado de abstracción dependiendo del propósito de la simulación. Existen implementaciones de modelos para el nivel de red, se pueden encontrar en los paquetes,

de.tud.kom.p2psim.impl.network.simple

de.tud.kom.p2psim.impl.network.gnp

Sobre la capa de red está la **capa de transporte.** Permite la visibilidad extremo a extremo entre dos nodos. Encapsula los detalles de la transferencia de datos entre nodos. El objetivo de los protocolos de este nivel es que se mantenga una cierta calidad en la comunicación entre extremos independientemente del estado de la red. Es importante poder proporcionar multiplexación de conexiones en el nivel de red, permitiendo que varias aplicaciones ejecutándose en la misma máquina, puedan utilizar recursos de red y comunicarse con sus entidades homólogas en otras máquinas.

Es muy interesante también en cuanto a la capa de red, el tipo de servicio que desea nuestra comunicación. Existen varios tipos de servicios proporcionados dependiendo de las características de los datos a transmitir:

- Orientado a conexión y fiable.
- No orientado a conexión y fiable.
- No orientado a conexión y no fiable.
- Orientado a conexión y no fiable.

El API de transporte debe permitir establecer la categoría de servicio que se necesita y simular su comportamiento en una red real. Las interfaces de este nivel se condensan en el paquete:

de.tud.kom.p2psim.api.transport

así como una extensión en el paquete,

de.tud.kom.p2psim.impl.transport

La **capa de overlay** es la siguiente. Una overlay es una red construida sobre una red existente. Es la red de nodos que se establece por encima de la red IP a nivel de aplicación. En este nivel se encuentran los protocolos Chord, Kademlia... Hay distintas interfaces para dar soporte a overlays centralizadas y descentralizadas donde podemos encontrar a su vez estructuradas y no estructuradas.

Las distintas overlays que se crean en esta capa de diferencian en cuanto a estructura y funcionalidad que ofrecen. A menudo se utilizan en distintas áreas de aplicación.

El paquete en el que se encuentran las clases es,

de.tud.kom.p2psim.api.overlay

Existe un conjunto de paquetes que son implementaciones de overlays. Todos respetan las interfaces del paquete anterior y hacen uso de las clases bases pero cada implementación tiene sus detalles propios. Estos paquetes son:

de.tud.kom.p2psim.impl.overlay.dht.chord

de.tud.kom.p2psim.impl.overlay.dht.kademlia

La **capa de aplicación** contiene componentes que permiten funcionar a las aplicaciones. Las aplicaciones son los procesos que corren en la parte alta de la pila de protocolos. En general, representan las implementaciones de los clientes. Los paquetes utilizados en este caso son,

de.tud.kom.p2psim.api.application

de.tud.kom.p2psim.impl.application

La última capa, la **capa de usuario**, captura las acciones realizadas por el usuario para modelar diferentes comportamientos.

Todos estos niveles funcionales se combinan e incorporan en un "host" que actúa como punto final en una red peer-to-peer.

El marco de simulación no permite la transferencia de datos directamente desde un nivel de un nodo al mismo nivel de otro nodo. Cada nivel pasa los datos y la información de control al nivel inferior hasta que se llega al "motor de la simulación" (el nivel que hay por debajo de la capa de red).

Cada vez que dos nodos quieren comunicarse, se crea un mensaje en la capa de aplicación y se pasa a la capa de transporte. La capa de transporte introduce información adicional al mensaje. Esta información es necesaria para que el nivel de aplicación del nodo destinatario pueda obtener el mensaje correctamente. La capa de red pasa el mensaje al motor de la simulación. En el nodo destinatario, el mensaje va haciendo el recorrido inverso hasta llegar a la capa de aplicación.

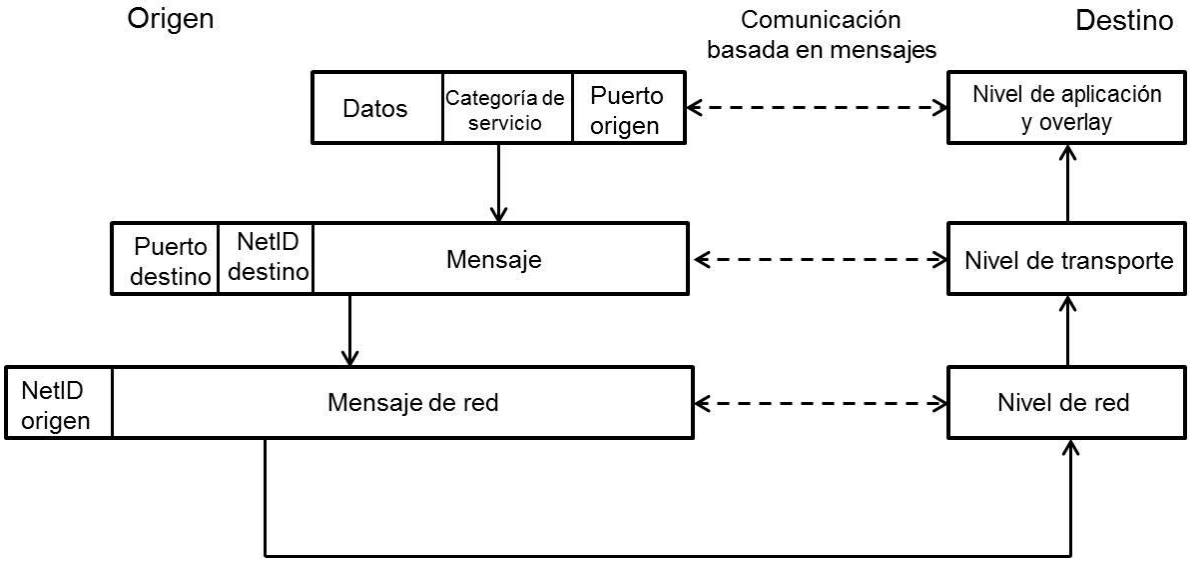


Fig. 52: Mensajes entre dos hosts

El simulador está diseñado de forma que sea de propósito general, es decir, no es para una arquitectura de red P2P específica. Debido a esto, existen una serie de acciones importantes y comunes a gran número de arquitecturas que se han implementado en el software de la arquitectura del simulador. Estos componentes describen generalmente funciones individuales bien definidas para ofrecer una funcionalidad requerida. Cualquier usuario puede realizar la implementación más adecuada de estas interfaces dependiendo de la red P2P con la que trabaja.

En la figura 52 se muestra la dependencia entre los principales paquetes que conforman el simulador y las relaciones que se establecen entre ellos.

A continuación, en la figura 53, se muestra una relación entre los distintos paquetes de los que hemos hablado y la implementación jerárquica realizada de la red Chord.

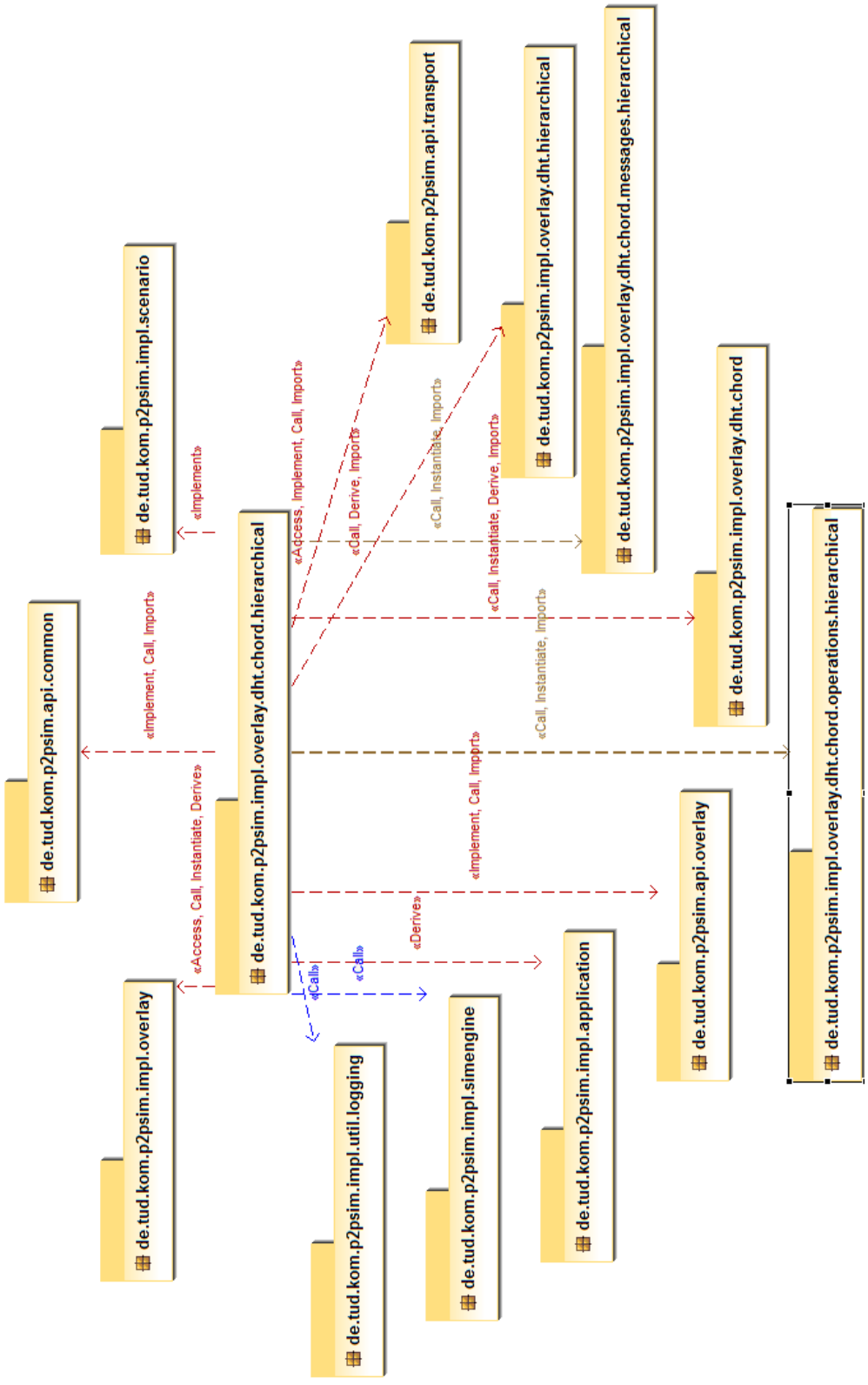


Fig. 53: Relación entre los paquetes del simulador

2. Como crear una overlay en PeerfactSim

El objetivo de este apartado es mostrar los primeros pasos que hay que dar para crear una overlay [17]. Después de esto, se podrán lanzar simulaciones de redes para evaluar la overlay.

Antes de empezar con la implementación, es necesario concretar las especificaciones de la overlay con la que se quiere trabajar. Esta especificación describe la estructura concreta de los componentes. Algunos elementos importantes a especificar son el nodo, el mecanismo de enrutado, que hacer cuando un nodo falla, como hacer “pings” para mantener la tabla de rutas actualizada, etc.

En una overlay, los equipos se llaman nodos. Son la parte principal de la implementación y están conectados virtualmente o lógicamente mediante enlaces o “links”. Las principales características de una overlay son:

- La red se construye sobre otra red ya existente.
- En la mayoría de los casos se define un espacio de direcciones, virtual o lógico, propio.

Se utiliza una técnica de enrutamiento para enviar mensajes a otros nodos.

3. Estructura básica de una overlay

Toda overlay tiene diferentes tipos de componentes que debemos implementar [17]**:**

- **Identificador de nodo**: Los nodos deben estar identificados inequívocamente, deben tener algún tipo de etiqueta lógica o virtual. Éstas, son calculadas o elegidas de forma aleatoria.

- **"Tarjeta de contacto" de nodo**: Es el componente que guarda el identificador de nodo y su dirección para que cualquier nodo pueda ponerse en contacto con él. Cada nodo guardará la "tarjeta de contacto" de una parte de los nodos de la red.

- **Identificación de mensajes y objetos de datos**: Todos los mensajes y los objetos de datos que se envían de un nodo a otro tienen un destino. Se sabe cual es el destino por el identificador de nodo que se ha visto antes. Los mensajes también tienen que tener un identificador.

- **Tabla de rutas**: Cada nodo tiene una lista con información sobre los nodos de una parte de la red. Dependiendo de las especificaciones de los protocolos se implementa la tabla de rutas de una manera u otra. Es necesario definir claramente cuál será la manera de almacenar los contactos de cada nodo.

- **Comandos**: Todos los comandos utilizados en la red están encapsulados en distintas clases así como todos los mensajes que se van a intercambiar en la red tienen que estar implementados.

- **Creación de los componentes para la simulación**: Para realizar la simulación de la red se tiene que dar al simulador la habilidad de crear correctamente los nodos y aplicaciones. Todos los componentes que tienen que ser creados por el simulador tienen una clase "factory class" que los crea.

4. Evaluación de la overlay

Los analizadores obtienen información acerca de la simulación de la red P2P. El simulador notifica a todos los analizadores sobre las partes más importantes de la simulación, como pueden ser la transmisión de mensajes o las operaciones de iniciación. Dependiendo del tipo de analizador, estos funcionaran automáticamente o no.

Anexo IV

Planificación y presupuesto

Durante el siguiente capítulo se expone de forma gráfica la planificación realizada así como todos los aspectos relativos al presupuesto del proyecto.

1. Planificación

Debido a la magnitud del proyecto, desde el principio del mismo se realizó una planificación para su posterior seguimiento durante todo el tiempo abarcado. Para realizar dicha tarea se ha utilizado un diagrama de Gantt que se puede ver a continuación junto con la lista de tareas.

	Nombre de tarea	Duración	Comienzo	Fin	Predecesora	Nombres de los recursos
1	**Fase inicial y mantenimiento**	**395 días**	**lun 12/10/09**	**vie 20/04/12**		**Jefe de proyecto**
2	Toma de requisitos	5 días	lun 12/10/09	vie 16/10/09		Jefe de proyecto
3	Planificación	5 días	lun 19/10/09	vie 23/10/09	2	Jefe de proyecto
4	Mantenimiento planificación	385 días	lun 26/10/09	vie 20/04/12	3	Jefe de proyecto
5	Reunión 1	1 día	lun 12/10/09	lun 12/10/09		Jefe de proyecto
6	Reunión 2	1 día	mar 20/10/09	mar 20/10/09		Jefe de proyecto
7	Reunión 3	1 día	vie 30/10/09	vie 30/10/09		Jefe de proyecto
8	Reunión 4	1 día	vie 04/12/09	vie 04/12/09		Jefe de proyecto
9	Reunión 5	1 día	lun 11/01/10	lun 11/01/10		Jefe de proyecto
10	Reunión 6	1 día	vie 19/02/10	vie 19/02/10		Jefe de proyecto
11	Reunión 7	1 día	lun 01/03/10	lun 01/03/10		Jefe de proyecto
12	Reunión 8	1 día	vie 05/03/10	vie 05/03/10		Jefe de proyecto
13	Reunión 9	1 día	lun 19/04/10	lun 19/04/10		Jefe de proyecto
14	Reunión 10	1 día	mié 23/06/10	mié 23/06/10		Jefe de proyecto
15	Reunión 11	1 día	lun 06/09/10	lun 06/09/10		Jefe de proyecto
16	Reunión 12	1 día	vie 14/01/11	vie 14/01/11		Jefe de proyecto
17	Reunión 13	1 día	lun 11/04/11	lun 11/04/11		Jefe de proyecto
18	Reunión 14	1 día	vie 02/09/11	vie 02/09/11		Jefe de proyecto
19	Reunión 15	1 día	jue 15/12/11	jue 15/12/11		Jefe de proyecto
20	**Fase de análisis**	**99 días**	**lun 26/10/09**	**vie 29/10/10**		**Analista;Programador**
21	Estudio redes P2P	9 días	lun 26/10/09	jue 05/11/09	3	Analista
22	Estudio Chord	22 días	vie 06/11/09	jue 08/04/10	21	Analista[33%];Programador[66%]
23	Estudio Kademlia	24 días	mar 20/04/10	vie 21/05/10	22	Analista[33%];Programador[66%]
24	Estudio PeerfactSim.KOM	25 días	lun 24/05/10	vie 01/10/10	23	Analista[33%];Programador[66%]
25	Estudio redes P2P jerárquicas	10 días	lun 04/10/10	vie 15/10/10	24	Analista
26	Estudio y decisión de tecnología	10 días	lun 18/10/10	vie 29/10/10	25	Analista
27	**Fase de diseño**	**53 días**	**lun 01/11/10**	**vie 25/02/11**		**Analista**
28	Diseño red jerárquica	43 días	lun 01/11/10	vie 11/02/11	26	Analista
29	Diseño programas auxiliares	10 días	lun 14/02/11	vie 25/02/11	28	Analista
30	**Fase de implementación**	**103 días**	**lun 28/02/11**	**vie 25/11/11**		**Programador**
31	Instalación y preparación de herramientas	5 días	lun 28/02/11	vie 04/03/11	29	Programador
32	Implementación H-Chord	78 días	lun 07/03/11	vie 28/10/11	31	Programador
33	Implementación programas auxiliares	20 días	lun 31/10/11	vie 25/11/11	32	Programador
34	**Fase de simulación y verificación**	**62 días**	**lun 28/11/11**	**vie 09/03/12**		**Analista;Programador**
35	Simulaciones	35 días	lun 28/11/11	mié 01/02/12	33	Programador
36	Estudio de datos	27 días	jue 02/02/12	vie 09/03/12	35	Analista
37	**Fase de documentación**	385 días	lun 26/10/09	vie 20/04/12		Programador[50%];Analista[50%]

Fig. 54: Lista de tareas

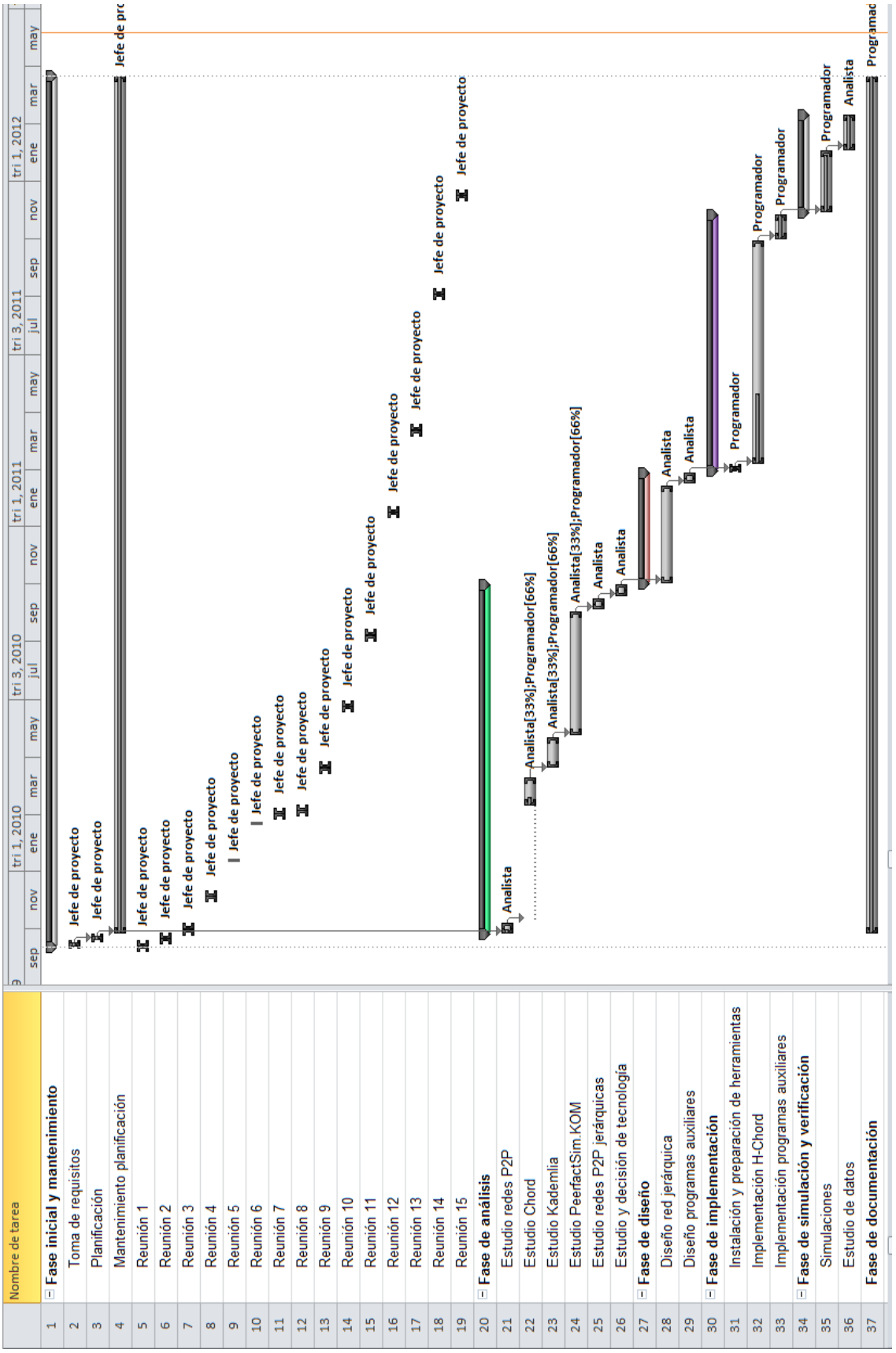


Fig. 55: Diagrama de Gantt

2. Presupuesto

A continuación se detalla el presupuesto del proyecto, especificando el gasto de personal, software y hardware generado durante el desarrollo. Para realizar el presupuesto se utiliza la plantilla proporcionada por la Universidad Carlos III de Madrid que se puede ver en el apartado 3 de este anexo. Los siguientes apartados son un desglose que se utilizarán para facilitar la lectura de los datos.

2.1. Coste de personal

Se ha fijado una *jornada de cuatro horas diarias* y se ha supuesto un esfuerzo continuo, aunque la línea de carga ha sido irregular a lo largo de todo el proceso. La solución tomada, por tanto, refleja una media real de las horas invertidas en el desarrollo del proyecto al completo.

A partir de la planificación expuesta en el apartado anterior es posible calcular fácilmente el número de horas dedicadas al proyecto, ascendiendo el total a 1440 horas durante casi veinticuatro meses de trabajo con la siguiente distribución entre fases.

- **Fase inicial y mantenimiento**: 40horas + 385 días *5 minutos + 15 * 1h/reunión = 87,08 horas.
- **Fase de análisis**: 99 días * 3,45 horas/día = 371,5 horas de las cuales el analista ha dedicado 193,72 y el programador 177,78.
- **Fase de diseño**: 53 días * 3,45 horas/día = 199,63 horas Todas llevadas a cabo por el analista
- **Fase de implementación**: 103 días * 3,45 horas/día = 386,25 horas Todas ellas asignadas al programador.

- **Fase de simulación y verificación:** 62 días * 3,45 horas/día = 232,5 horas de las cuales el analista ha dedicado 101,25 y el programador 131,25.

- **Documentación**: 385 días * 15 minutos/día = 96,25 horas de las cuales tanto el analista como el programador han dedicado 48,12 horas.

Los distintos cargos necesarios para realizar dichas tereas son los siguientes:

- **Jefe de proyecto**: realiza las tareas de planificación, tanto de las tareas como de los recursos y es el encargado de la relación con el cliente. Total de horas empleadas: 87,08.

- **Analista**: Realiza un estudio del mercado, tecnología, herramientas y características de la red y se encarga de realizar el diseño y documentarlo. Total de horas empleadas: 542,72.

- **Programador**: Implementa el diseño realizado por el analista y escribe la parte más técnica de la documentación. Total de horas empleadas: 743,4.

Teniendo en cuenta la planificación realizada en *jornadas de cuatro horas*, tenemos que el valor de un hombre/mes es igual a ochenta horas de trabajo, habiéndose hecho los cálculos de la tabla 2 a partir de este dato.

Categoría	Dedicación hombre mes	Coste hombre mes (€) *	Coste total (€)
Jefe de proyecto	1,5	3.200 (40€/hora)	4.800
Analista	7	2.560 (32€/hora)	17.920
Programador	9,5	2.000 (25€/hora)	19.000
***1 hombre/mes =80 horas**			41.720

Tabla 2: Gasto de personal

2.2. Coste de Hardware

La necesidad del hardware para llevar a cabo el desarrollo de un proyecto es indispensable. Aunque parte de los elementos utilizados son propiedad del autor del proyecto, se han realizado algunas inversiones cuyos precios y coste total imputable se detallan a continuación. Este cálculo se ha realizado conforme a lo indicado en la plantilla de presupuesto referenciada anteriormente.

Descripción	Coste (€)	% de uso dedicado	Dedicación meses	Periodo de depreciación	Coste imputable (€)
HP Presario SR5 340ES Quad Q6600 4G 500G	378	100	24	60	148,30
Ratón y teclado Compaq	20	100	24	60	7,85
Monitor TFT Sony SDM-HX73	223	100	24	60	87,49
Pen drive 8 Gb	12	100	24	60	4,71
PC i7-2600, 12 Gb, RAM	729	100	24	60	291,6
					539,95 €

Tabla 3: Coste de Hardware

Fórmula de cálculo de la amortización: $\frac{A}{B} * C * D$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado

B = periodo de depreciación (60 meses)

C = coste del equipo (sin IVA)

D = % del uso que se dedica al proyecto (habitualmente 100%)

2.3. Coste de Software

Durante el proyecto, se han utilizado diferentes aplicaciones y herramientas para generar tanto el código de la aplicación como toda la documentación. Por ello se incluyen dichas licencias dentro del presupuesto.

La mayoría del software utilizado es software libre por lo que su coste es cero. Sólo han sido necesarias licencias de dos aplicaciones.

Descripción	Coste (€)
Ubuntu Hardy Heron	0
Eclipse Galileo	0
Citrix XenCenter	0
OpenOffice	0
Microsoft Office Project	119
	119 €

Tabla 4: Coste de Software

2.4. Resumen de costes

La siguiente tabla un resumen de todos los costes anteriormente detallados incluyendo una tasa de costes indirectos del 20% que será el presupuesto final alcanzado mediante el uso de la plantilla del presupuesto. A parte se incluye un coste del 18% en concepto de impuestos sobre el valor añadido.

Descripción	Coste (€)
Personal	41.720
Hardware	540
Software	119
Subcontratación de tareas	0
Costes indirectos (20%)	8.476
Total	50.855
Total + IVA (18%)	60.009 €

Tabla 5: Resumen de costes

El presupuesto total de este proyecto asciende a la cantidad de SESENTA MIL NUEVE EUROS.

Leganés a 20 de MAYO de 2012

Fdo. Sandra Marco Espinel

3. Plantilla de presupuesto

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1.- Autor:
Sandra Marco Espinel
2.- Departamento:
Ingeniería Telemática
3.- Descripción del Proyecto:

- Titulo	**Simulacion de redes jerárquicas P2P**
- Duración (meses)	**24**
Tasa de costes Indirectos:	20%

4.- Presupuesto total del Proyecto (valores en Euros):
60,813.00 Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F. (no rellenar - solo a titulo informativo)	Categoría	Dedicación (hombres mes) [a]	Coste hombre mes [b]	Coste (Euro)	Firma de conformidad
Sandra Marco Espinel		Jefe de proyecto	1.5	3,200.00	4,800.00	
Sandra Marco Espinel		Analísta	7	2,560.00	17,920.00	
Sandra Marco Espinel		Programador	9.5	2,000.00	19,000.00	
					0.00	
					0.00	
		Hombres mes	**18**	**Total**	41,720.00	

[a] 1 Hombre mes = 80 horas.
[b] Jefe de proyecto 40€/h, analista 32€/h, programador 25€/h

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d]
HP Presario SR5 340ES Quad Q66	378.00	100	24	60	148.30
Ratón y teclado Compaq	20.00	100	24	60	7.85
Monitor TFT Sony SDM-HX73	223.00	100	24	60	87.49
Pen drive 8 Gb	12.00	100	24	60	4.71
PC i7-2600, 12 Gb, RAM	729.00	100	24	60	291.60
				Total	539.95

[d] Fórmula de cálculo de la Amortización:

$$\frac{A}{B} x C x D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
	Total	0.00

OTROS COSTES DIRECTOS DEL PROYECTO

Descripción	Empresa	Costes imputable
Ubuntu Hardy Heron		0.00
Eclipse Galileo		0.00
Microsoft Office Project		119.00
Citrix XenCenter		0.00
OpenOffice		0.00
	Total	119.00

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	41,720
Amortización	540
Subcontratación de tareas	0
Otros costes directos	119
Costes Indirectos	8,476
Total	50,855

Bibliografía

1. **Sanz, M.A.** *Fundamentos históricos de Internet en Europa y España.*

2. **Afanador J. A., Ribero D. F., Ulloa G. E.** *Redes P2P y enrutamiento en capa de aplicación.* 2006.

3. **Stoica, I., Morris, R., Karger, K., Kaashoek, F., Balakrishnan, H.** *Chord: A scalable peer-to-peer lookup service for internet applications.* MIT laboratory for computer science. 2001.

4. **Stiller, J. Mischke and B.** *Peer-to-peer overlay network management through agile.* 2003. págs. 337-350.

5. **Eriksson, P.** *H-Chord: A hierarchical resource indexing scheme for dynamic environments.* Royal Institute of technology, Stockholm. 2005.

6. **Maymounkov, P., Mazières, D.** *Kademlia: A Peer-to-peer information system based on the XOR metric.*

7. **González Sánchez, R.** *Implementación de redes overlay jerárquicas usando el protocolo P2PP.* 2009.

8. **Garces-Erice, L., y otros.** *Hierarchical p2p systems.* 2003.

9. **Millán Tejedor, R.J.** *Domine las redes P2P : "Peer to Peer" orígenes, funcionamiento y legislación del.* s.l. : Creaciones Copyright, 2006.

10. **Steiner, M., En-Najjary, T. and W.Biersack, E.** *A global view of KAD.* Institute Eurecom, France : s.n.

11. **Martínez Yelmo, I.** *Design and evaluation of interconnecting structured peer-to-peer networks with a hierarchical topology.* 2009.

12. **Leslie, M., Davies, J. and Huffman, T.** *A comparasion of replication strategies for reliable decentralised storage.* Diciembre 2006.

13. **Lambing, D.** *Chord network simulation.* 2008.

14. **Martinez-Yelmo, I., Bikfalvi, A., Guerrero, C., Rumín, R.C. and Mauthe, A.**

Enabling global multimedia distributed services based on hierarchical DHT overlay networks. 2008.

15. http://www.citrix.es/Productos_y_Soluciones/Productos/XenServer/Presentation/. [En línea]

16. **Kaune, Sebastian.** *PeerfactSim.KOM - A simulator for large scale peer-to-peer systems.* Germany : s.n., 2007.

17. **Esen, E.** *How to create an overlay in PeerfactSim.KOM.* 2008.

18. **Spori, B.** *Implementation of the Kademlia distributed hash table.* Swiss federal institute of technology Zurich. 2006.

19. [En línea] es.wikipedia.org.

20. **M.S. Artigas, P.G. López, J.P. Ahullo, and A.F.G. Skarmeta.** *Cyclone: a novel design schema for hierarchical dhts.* 2005. págs. 49-56.

21. **Cortell-Albert, J.** *El futuro de redes P2P en entornos corporativos.*

22. **I.Stoica, R. Morris, D. Liben-Nowell, DR Karger, MF Kaashoek, F. Dabek, and H. Balakrishanan.** *Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Application.*

23. **Keong Lua, E., Crowcroft, J., Pias, M., Sharma, R., Lim, S.** *A survey and comparison of Peer-to-peer Overlay network schemes.* 2004.

24. **Ktari, S., Hecker, A., Labiod, H.** *Power-law chord arquitecture in P2P overlays.*

25. **Shelke, S., Shirodkar, S.** *Dsitributed hash-based lookup for Peer-to-Peer systems.* 2006.

26. **Terrell, J.** *A survey of theorical issues in Peer-to-Peer Systems.*

27. **Prasanna, G., Gudammi, K. y García Molina, H.** *Canon en g major: Designing dhts with hierarchical structure.* 2004. págs. 263-272.

Glosario

AudioGalaxy Se utilizó como motor de búsqueda de archivos mp3 sobre servidores FTP primero y posteriormente paso a se runa aplicación P2P destinada al intercambio de música entre usuarios a través de Internet.

BitTorrent Actualmente, uno de los protocolos más utilizados para el intercambio de archivos peer-to-peer en internet.

Bootstrap Elemento encargado de proporcionar a los nuevos peers información sobre los nodos que ya forman parte de la red.

Bucket En kademlia, listas que tiene los nodos con información sobre otros nodos.

CAN Content Addressable Network. Protocolo de búsqueda distribuida en redes peer-to-peer

Chord Protocolo de búsqueda distribuida en redes peer-to-peer.

Churn Número medio de nodos que abandonan la red en un determinado intervalo de tiempo.

DHT Distributed Hash Table. Clase de sistemas distribuidos descentralizados que proveen un servicio de búsqueda similar al de las tablas de hash, donde pares (clave, valor) son almacenados en el DHT, y cualquier nodo participante puede recuperar de forma eficiente el valor asociado con una clave dada.

eDonkey Red de intercambio de archivos P2P.

Finger table Tabla de rutas que cada nodo tiene en Chord.

Freenet Red de distribución de información descentralizada. Trabaja por medio de la puesta en común del ancho de banda y el espacio de almacenamiento de los ordenadores que componen la red, permitiendo a sus usuarios publicar o recuperar distintos tipos de información anónimamente

Gnutella Protocolo de red de distribución de archivos P2P sin servidor central.

Hash Una función de hash proporciona una clave, que representa un contenido, de manera casi inequívoca.

ID Identificador.

IP Internet Protocol. Protocolo no orientado a conexión, usado tanto por el origen como por el destino para la comunicación de datos, a través de una red de paquetes conmutados no fiable.

Kademlia Protocolo de búsqueda distribuida en redes peer-to-peer.

Key Clave que representa a un dato o un nodo dentro de una red.

Napster Napster fue un servicio de distribución de archivos de música en formato MP3, la primera gran red P2P de intercambio.

Node	Cada uno de los elementos que participan en la red overlay.
Overlay	Red de nodos conectados de forma virtual.
P2P	Peer-to-peer.
Peer	Cada uno de los elementos que participan en la red overlay.
PeerfactSim.KOM	Simulador elegido para lanzar las simulaciones.
Ping	Packet Internet Groper. Utilidad que comprueba, un una red de ordenadores, el estado del enlace entre el nodo local y otro nodo de la red.
Query	Búsqueda en una red overlay.
Skype	Software que permite comunicaciones de texto, voz y vídeo sobre Internet (VoIP). Fue desarrollado en 2003 por el danés Janus Friis y el sueco Niklas Zennström.
SopCast	Programa para ver y emitir canales de televisión y video a través de internet basado en P2P.
Value	Dato almacenado en una red P2P
VoIP	Voz sobre IP. Conjunto de recursos que permiten que la señal de voz viaje a través de Internet empleando el protocolo IP